Btrieve Programmer's Reference

d i s c l a i m e r   PERVASIVE SOFTWARE INC. LICENSES THE SOFTWARE AND
DOCUMENTATION PRODUCT TO YOU OR YOUR COMPANY SOLELY ON
AN "AS IS" BASIS AND SOLELY IN ACCORDANCE WITH THE TERMS AND
CONDITIONS OF THE ACCOMPANYING LICENSE AGREEMENT.
PERVASIVE SOFTWARE INC. MAKES NO OTHER WARRANTIES
WHATSOEVER, EITHER EXPRESS OR IMPLIED, REGARDING THE
SOFTWARE OR THE CONTENT OF THE DOCUMENTATION; PERVASIVE
SOFTWARE INC. HEREBY EXPRESSLY STATES AND YOU OR YOUR
COMPANY ACKNOWLEDGES THAT PERVASIVE SOFTWARE INC. DOES
NOT MAKE ANY WARRANTIES, INCLUDING, FOR EXAMPLE, WITH
RESPECT TO MERCHANTABILITY, TITLE, OR FITNESS FOR ANY
PARTICULAR PURPOSE OR ARISING FROM COURSE OF DEALING OR
USAGE OF TRADE, AMONG OTHERS.

t r a d e m a r k s   Btrieve and XQL are registered trademarks of Pervasive Software Inc.

Built on Btrieve, Built on Scalable SQL, Client/Server in a Box, DDF Ease
InstallScout, MicroKernel Database Engine, MicroKernel Database Architecture,
Navigational Client/Server, Pervasive.SQL, Scalable SQL, Smart Components,
Smart Component Management, Smart Naming, SmartScout, and Xtrieve PLUS
are trademarks of Pervasive Software Inc.

Microsoft, MS-DOS, Windows, Windows NT, Win32, Win32s, and Visual Basic
are registered trademarks of Microsoft Corporation.

Windows 95 is a trademark of Microsoft Corporation.

NetWare and Novell are registered trademarks of Novell, Inc.

NetWare Loadable Module, NLM, Novell DOS, Transaction Tracking System,
and TTS are trademarks of Novell, Inc.

All company and product names are the trademarks or registered trademarks of
their respective companies.

Contents

Tables

About This Manual

This manual contains information about how to develop applications that use the Btrieve navigational data management system. Btrieve is designed for high-performance data handling and improved programming productivity.

The functions described in this manual are available in the modules installed by the Programming Interfaces installation option.

You may also puarchase the Programmer's Suite. The Programmer's Suite product provides:

♦ all of the interfaces described in this manual

♦ a 5-user version of both the NetWare and Windows NT server engines licensed for development and testing use only

♦ the complete set of Pervasive documentation in hardcopy

Who Should Read This Manual

This manual provides information for developers who are using Btrieve to develop applications for the OS/2, NetWare, Windows, Windows NT, and Windows 95 operating environments.

Pervasive Software would appreciate your comments and suggestions about this manual. Please complete the User Comments form that appears at the end of this manual, and fax or mail it to Pervasive Software, or send email to docs@pervasive.com.

Organization

◆ Chapter 1—"Introduction to Btrieve APIs"

This chapter provides a brief overview of the Btrieve functions and their parameters. This chapter also summarizes the Btrieve operations.

◆ Chapter 2—"Btrieve Operations"

This chapter documents the Btrieve operations.

◆ Appendix A—"Language Interfaces"

This appendix provides specific information about the language interfaces.

◆ Appendix B—"Data Types"

This appendix provides information about the data types Btrieve supports for keys.

◆ Appendix C—"Quick Reference of Btrieve Operations"

This appendix summarizes the Btrieve operations in order of operation number.

Conventions

Unless otherwise noted, command syntax, code, and code examples use the following conventions:

Case

Commands and reserved words typically appear in uppercase letters. Unless the manual states otherwise, you can enter these items using uppercase, lowercase, or both. For example, you can type MYPROG, myprog, or MYprog.

[ ]

Square brackets enclose optional information, as in [*log_name*]. If information is not enclosed in square brackets, it is required.

|

A vertical bar indicates a choice of information to enter, as in [*file name* | @*file name*].

< >

Angle brackets enclose multiple choices for a required item, as in `/D=<5│6│7>`.

variable

Words appearing in italics are variables that you must replace with appropriate values, as in *file name*.

…

An ellipsis following information indicates you can repeat the information more than one time, as in [*parameter* …].

::=

The symbol ::= means one item is defined in terms of another. For example, a::=b means the item *a* is defined in terms of *b*.

c h a p t e r **1** # Introduction to Btrieve APIs

The Btrieve navigational database management system is designed for high-performance data handling and improved programming productivity. Btrieve operations allow your application to retrieve, insert, update, or delete records either by key value, or by sequential or random access methods.

The Programming Interfaces provide compatibility with the following programming languages and development environments:

- ◆ Borland C/C++
- ◆ Borland Delphi
- ◆ Micro Focus COBOL
- ◆ Microsoft Access
- ◆ Microsoft Visual Basic
- ◆ Microsoft Visual C++
- ◆ Watcom C/C++

This chapter discusses the following topics:

- ◆ "Btrieve Functions"
- ◆ "Btrieve Function Parameters"
- ◆ "Summary of Btrieve Operations"
- ◆ "Sequence of Events in Performing a Btrieve Operation"

Btrieve Functions

Btrieve offers a single-function API, in which most program actions are determined by an operation code parameter, rather than a function name. You should choose the API for your application based on whether you are most interested in cross-platform portability of code or the best possible performance on a particular platform.

Your Btrieve application should never perform any standard I/O against a data file. Your application should perform all file I/O using a Btrieve function.

Following are the Btrieve functions.

Table 1-1   Btrieve Functions

Function	Operating Systems	Description
BTRV BTRVID	All	Use for complete code portability between operating systems. For most developers, this advantage offsets a very slight performance decrease.
BTRCALL BTRCALLID	OS/2, Windows 3.x, Windows NT, and Windows 95 only	Use these functions to achieve a slight performance increase or backward compatibility with existing source code.
BTRCALL32 BTRCALLID32	For 32-bit OS/2 applications only	

To find the language-specific syntax required when calling a Btrieve function, refer to Appendix A, "Language Interfaces".

BTRV Function

BTRV allows an application to make Btrieve calls. All the language interface modules provided with the Programming Interfaces installation option support the BTRV function. In OS/2, Windows 3.*x*, Windows NT, and Windows 95, the BTRV function actually calls the BTRCALL function. However, BTRV is the preferred function because of the platform independence it provides.

BTRVID Function

BTRVID allows an application to make a single Btrieve call that contains a clientID parameter, which the application can control. An application can use BTRVID to assign itself more than one client identity to Btrieve and to execute operations for one client without affecting the state of the other clients. For more information, refer to "Client ID".

In OS/2, Windows 3.*x*, Windows NT, and Windows 95, the BTRVID function actually calls another function. In 16-bit applications, it calls the BTRCALLID function. In 32-bit applications, it calls the BTRCALLID32 function (OS/2) or the BTRCALLID function (Windows NT and Windows 95). However, in both cases, BTRVID is the preferred function because of the platform independence it provides.

In DOS applications, you must load the DOS Requester with the appropriate /T value. Set /T to equal the number of client IDs you use in the application. For more information about the DOS Requester, refer to *Getting Started*.

BTRCALL Function

BTRCALL is the preferred function if your application accesses only local files and uses only the OS/2 or Win32 workstation engines. For OS/2, the BTRCALL function is a 16-bit function. For Windows NT and Windows 95, the BTRCALL function is a 32-bit function.

BTRCALL32 Function

The BTRCALL32 function is the same as BTRCALL function, except that BTRCALL32 is a 32-bit function.

BTRCALLID Function

Use the BTRCALLID function if you need client-level control and your application operates only in the OS/2, Windows 3.x, Windows NT and Windows 95 environments.

This function is similar to the BTRVID function, except that it does not call an intermediate function. For Windows NT and Windows 95, BTRCALLID is a 32-bit function.

BTRCALLID32 Function

The BTRCALLID32 function is the same as the BTRCALLID function, except that the BTRCALLID32 is a 32-bit function.

Obsolete Functions

The following historical functions are supported to maintain compatibility with applications written for previous Btrieve releases:

- ◆ BTRCALLBACK

- ◆ BTRVINIT

- ◆ BTRVSTOP

- ◆ RQSHELLINIT

- ◆ WBRQSHELLINIT

- ◆ WBTRVINIT

- ◆ WBTRVSTOP

- ◆ BRQSHELLINIT

While these functions are now obsolete, older applications that call these functions will still run with 6.15 and later MicroKernels. (No additional functions have been made obsolete in Btrieve 7.0.)

Btrieve Function Parameters

You must provide all parameters on every call; however, the MicroKernel does not use every parameter on every operation. In some cases, the MicroKernel ignores their value. In general, different parameters can be sent and returned for each operation. Chapter 2, "Btrieve Operations" provides a detailed description of the parameters that are relevant for each Btrieve operation.

Note

C developers: Refer to BTITYPES.H for a description of the platform-independent data types and pointers used in the C language interface.

The parameters to the Btrieve functions are as follows:

- ◆ Operation Code

- ◆ Status Code (BASIC and COBOL only)

- ◆ Position Block

- ◆ Data Buffer

- ◆ Data Buffer Length

- ◆ Key Buffer

- ◆ Key Number

- ◆ Client ID (BTRVID and BTRCALLID functions only)

- ◆ Key Length (BTRCALL, BTRCALLID, BTRCALL32, and BTRCALLID32 functions only)

Operation Code

The Operation Code parameter determines what action is performed by the Btrieve function. For example, the operation may read, write, delete, or update one or more records. Your application must specify a valid Operation Code on every Btrieve call. The MicroKernel never changes the Operation Code. The value of the variable you specify can be any one of the legal Btrieve Operation Codes described in Chapter 2, "Btrieve Operations."

Note

C developers: The variable you specify must be BTI_WORD (an unsigned short integer).

Status Code

Btrieve returns status codes as signed integers. In most programming environments, the status code is the return value of the Btrieve function call. However, some BASIC and

COBOL language interfaces require a Status Code parameter. This parameter is a 2-byte integer containing a coded value that indicates whether any errors occurred during the operation. After a Btrieve call, the application must always check the value of the status variable to determine if the call was successful.

Pervasive.SQL components return status codes from calls to their APIs. When you write to these APIs, you should provide handling for three conditions:

♦ API Success

♦ Anticipated API failure

♦ Unanticipated API failure

Following is a C code example that handles all three conditions.

```
status = BTRVID(B_VERSION, posBlock1, &versionBuffer,
    &dataLen, keyBuf1, keyNum, (BTI_BUFFER_PTR) &clientID);
if (status == B_NO_ERROR)
{
    /* continue normal operation */
    status = BTRVID(...);
}
else if (status == B_RECORD_MANAGER_INACTIVE)
{
    /* handle known error */
    printf("Btrieve Get Version() returned
        B_RECORD_MANAGER_INACTIVE\n");
}
else
{
    /* unanticipated error */
    printf("Btrieve Get Version() returned %d\n", status);
} /* end if-else */
```

By following this method of status code handling, you can help Pervasive Software ensure your application's future stability. Pervasive Software encourages and incorporates developer feedback in order to continuously improve our products. For example, in Pervasive.SQL 7, Status Code 20 has been differentiated into several additional status codes. Applications that handle status information as demonstrated here can accept such enhancements gracefully. (For more information about the differentiation of status codes, refer to *Status Codes and Messages*.)

Position Block

The Position Block parameter is the address of a 128-byte array that the MicroKernel uses to store file I/O structures and the positioning information associated with an Open (0) operation. Each time your application opens a file, it must allocate a unique Position Block.

Btrieve initializes the Position Block when your application performs the Open operation, then references and updates it during file operations. Therefore, your application must specify the same Position Block on all subsequent Btrieve operations for the file.

> **Note**
>
> Do not write to the Position Block. Doing so could result in a lost position error, other errors, or damage to the file.

When you open more than one file at a time, the MicroKernel uses the Position Block to determine which file a particular call is for. Similarly, when you open the same file more than once, the MicroKernel uses a different Position Block for each separate Open operation. Likewise, the MicroKernel uses a different Position Block for each separate client that opens the same file. Multiple clients cannot share position blocks.

Data Buffer

Your application transfers data to and from a file using the Data Buffer. The information passed to or from the MicroKernel in the Data Buffer depends on which Btrieve operation is being performed. Frequently, the Data Buffer contains one or more records that your application is transferring to or from a file. However, depending on the Btrieve operation, the Data Buffer can contain other information, such as file or key specifications, MicroKernel version information, and so on.

Be sure to allocate a large enough Data Buffer to accommodate the longest record in your file. If your Data Buffer Length parameter specifies a value smaller than the size of your Data Buffer, Btrieve modification operations may destroy data following the Data Buffer.

Data Buffer Length

For any operation that requires a Data Buffer, your application must pass a variable that indicates the size (in bytes) of the Data Buffer, which should be large enough to contain data that the operation returns.

> **Note**
>
> *BASIC developers:* Your application must pass the Data Buffer Length parameter ByRef as an integer.
>
> *C, COBOL, and Pascal developers:* Your application must pass the Data Buffer Length parameter as a pointer to a 2-byte unsigned integer.

When you are inserting records into or updating a file with variable-length records, the Data Buffer Length should equal the record length specified when you first created the file, plus the number of characters included beyond the fixed-length portion. When you are retrieving variable-length records, the Data Buffer Length should be large enough to

accommodate the longest record in the file. If a record is longer than 64 KB, you must use a chunk operation to operate on a portion of the record.

The MicroKernel uses the Data Buffer Length parameter to determine how much space is available in the Data Buffer. If you pass a Data Buffer Length that is longer than the Data Buffer you have allocated, you may cause the MicroKernel to overwrite memory. The Data Buffer Length should always represent the size of the allocated Data Buffer.

Key Buffer

Your application must pass the Key Buffer parameter on every Btrieve operation, even if that operation does not use a Key Buffer. Depending on the operation, your application may set the data in the Key Buffer, or the Btrieve function may return it.

> **Note**
>
> *BASIC developers:* Your application must pass the Key Buffer as a string. If the key value is an integer, your application should convert it to a string using the MKI$ statement before calling the Btrieve function. If a key consists of two or more segments, you must concatenate them into a single string variable and pass the variable as the Key Buffer.
>
> The MicroKernel returns an error if the string variable passed as the Key Buffer is shorter than the key's defined length. If your application's first call does not require initialization of the Key Buffer, assign the string variable the value SPACE$($x$), where x represents the key's defined length. Until your application assigns some value in BASIC to the string variable, it has a length of 0.
>
> *C developers:* Your application must pass the Key Buffer as the address of a variable containing the key value. The file BTITYPES.H defines the Key Buffer as a VOID pointer (BTI_VOID_PTR). Your application can then define the Key Buffer type as needed.

COBOL developers: Your application must pass the Key Buffer as a record variable. If the key consists of two or more segments, list them in the correct order as individual fields under an 01 level record. Then you can pass the entire record as the Key Buffer.

Pascal developers: Your application must pass the Key Buffer as a variable containing a key value. If a key consists of two or more segments, use a record structure to define the individual fields in the key.

In most environments, the MicroKernel cannot determine the Key Buffer length when an application makes a Btrieve call. Therefore, you must ensure that the buffer is at least as long as the key length you specified when you first created the key. Otherwise, Btrieve operations may destroy data stored in memory following the Key Buffer. It is best to always have a 256-byte Key Buffer.

Key Number

The information passed in the Key Number parameter depends on which operation is being performed. Most often, the Key Number contains a value that indicates which of up to 119 key (access) paths to follow for a particular operation. However, other information can be sent or returned in the Key Number parameter, such as a value indicating in what mode a file is to be opened.

For BTRV and BTRVID functions, the Key Number parameter is a 2-byte integer. For BTRCALL, BTRCALLID, BTRCALL32, and BTRCALLID32 functions, the Key Number parameter is a 1-byte signed CHARACTER (BTI_CHAR). In all functions, the Key Number parameter has a value range of 0 through 118. A Btrieve function never alters the Key Number parameter.

Client ID

The Client ID parameter is used only in the BTRVID and BTRCALLID functions. The Client ID parameter is the address of a 16-byte structure that allows the MicroKernel to differentiate among the clients on a computer. Use the following structure for the Client ID.

Table 1-2   Client ID Structure

Element	Length (bytes)	Description	
Filler	12	Initialize to 0.	
Service Agent ID	2	Identifies each instance of your application to the MicroKernel. This is a 2-character ASCII value. The value of this identifier must be greater than or equal to the ASCII value AA (0x41 0x41). The MicroKernel assumes special meaning for the following values:	
		0x4140 (@A)	Used internally.
		0xFFFF	Used internally.
		0x4952 (RI)	Used internally.
		0x4553 (SE) 0x4353 (SC) 0x4544 (DC) 0x4544 (DE) 0x5544 (DU)	Used to identify clients originated by Scalable SQL.
		0x5257 (WR)	Used by Btrieve Requesters.

Table 1-2   Client ID Structure *continued*

Element	Length (bytes)	Description
Client Identifier	2	Establishes a client's identity within the current instance of your application. The MicroKernel uses this unique identifier for concurrency and transaction-processing purposes.

Key Length

The Key Length parameter is used only in the BTRCALL, BTRCALLID, BTRCALL32, and BTRCALLID32 functions and is specific to the OS/2 and Windows 3.x operating systems.

While the Key Length parameter is currently reserved for internal use by the MicroKernel, if your application calls BTRCALL, BTRCALLID, BTRCALL32, or BTRCALLID32, it must pass the Key Length parameter as an unsigned char (BTI_BYTE), with a value of 255 (the maximum length of any key).

Summary of Btrieve Operations

Btrieve provides over 40 operations that you can call from your application program. The following tables summarize these operations. Refer to Chapter 2, "Btrieve Operations" for complete descriptions of the Btrieve operations. Refer to Appendix C, "Quick Reference of Btrieve Operations" for a summary of Btrieve operations ordered by operation code.

Session-Specific Operations

The following operations allow you to set or retrieve the current directory, shut down a workstation MicroKernel, retrieve the MicroKernel version number, terminate a client connection with the server MicroKernel, and begin, end, or abort a transaction. In applications that handle multiple clients, these operations are specific to the calling client.

Table 1-3   Session-Specific Operations

Operation	Code	Description
Set Directory	17	Changes the current directory.
Get Directory	18	Returns the current directory.
Stop	25	Terminates the workstation MicroKernel (not available for server-based MicroKernels).
Version	26	Returns the version number of the MicroKernel.
Reset	28	Releases all resources held by a client.

Table 1-3   Session-Specific Operations *continued*

Operation	Code	Description
Begin Transaction	19 1019	Marks the beginning of a set of logically related operations. Operation 19 begins an exclusive transaction. Operation 1019 begins a concurrent transaction.
End Transaction	20	Marks the end of a set of logically related operations.
Abort Transaction	21	Removes operations performed during an incomplete transaction.

File-Specific Operations

The following operations deal with a specific file, and therefore use the position block parameter to identify the file on which to operate. The file-specific operations are of three types:

- ◆ **File access and information.** These operations allow you to create a file, open and close a file, retrieve file statistics, set and clear the file's owner name, start or stop continuous operation mode on a file, unlock a file, and create and drop indexes on a file.

- ◆ **Data retrieval.** These operations allow you to retrieve a single record or a set of records given specified criteria. Btrieve supports data retrieval either by logical location in an index path or by physical location. For more information, refer to "Accessing Records" in the *Btrieve Programmer's Guide*.

  In addition, you can apply biases to the operation codes to control file and record locking in multi-client situations. For more information, refer to "Supporting Multiple Clients" in the *Btrieve Programmer's Guide*.

- ◆ **Data manipulation.** These operations allow you to insert, update, or delete data.

Table 1-4   File Access and Information Operations

Operation	Code	Description
Open	0	Makes a file available for access.
Close	1	Releases a file from availability.
Create	14	Creates a file with the specified characteristics.
Stat	15	Returns file and index characteristics, and number of records.
Stat Extended	65	Returns file names and paths of an extended file's components and reports whether a file is using a system-defined log key.
Set Owner	29	Assigns an owner name to a file.
Clear Owner	30	Removes an owner name from a file.
Continuous Operation	42	Server-based MicroKernels only. Places the specified file in or removes the file from continuous operation mode, for use in system backups.
Unlock	27	Unlocks a record or records.
Create Index	31	Creates an index.
Drop Index	32	Removes an index.

Table 1-5   Data Retrieval Operations

Operation	Code	Description
Index-Based (Logical) Data Retrieval		
Get Equal	5	Returns the first record in the specified index path whose key value matches the specified key value.
Get Next	6	Returns the record following the current record in the index path.
Get Previous	7	Returns the record preceding the current record in the index path.
Get Greater Than	8	Returns the first record in the specified index path whose key value is greater than the specified key value.
Get Greater Than or Equal	9	Returns the first record in the specified index path whose key value is equal to or greater than the specified key value.
Get Less Than	10	Returns the first record in the specified index path whose key value is less than the specified key value.
Get Less Than or Equal	11	Returns the first record in the specified index path whose key value is equal to or less than the specified key value.
Get First	12	Returns the first record in the specified index path.
Get Last	13	Returns the last record in the specified index path.
Get Next Extended	36	Returns one or more records that follow the current record in the index path. Filtering conditions can be applied.
Get Previous Extended	37	Returns one or more records that precede the current record in the index path. Filtering conditions can be applied.

Table 1-5   Data Retrieval Operations *continued*

Operation	Code	Description
Get Key	+50	Detects the presence of a key value in a file, without returning an actual record.
Get By Percentage	44	Returns the record located approximately at a position derived from the specified percentage value.
Find Percentage	45	Returns a percentage figure based on the current record's position in the file.
Non-Index-Based (Physical) Retrieval		
Get Position	22	Returns the position of the current record.
Get Direct/Chunk	23	Returns data from the specified portions (chunks) of a record at a specified position.
Get Direct/Record	23	Returns the record at a specified position.
Step Next	24	Returns the record from the physical location following the current record.
Step First	33	Returns the record in the first physical location in the file.
Step Last	34	Returns the record in the last physical location in the file.
Step Previous	35	Returns the record in the physical location preceding the current record.
Step Next Extended	38	Returns one or more successive records from the location physically following the current record. Filtering conditions can be applied.

Table 1-5   Data Retrieval Operations *continued*

Operation	Code	Description
Step Previous Extended	39	Returns one or more preceding records from the location physically preceding the current record. Filtering conditions can be applied.
Get By Percentage	44	Returns the record located approximately at a position derived from the specified percentage value.
Find Percentage	45	Returns a percentage figure based on the current record's position in the file.
Concurrency Control Biases (Add to the Appropriate Operation Code)		
Single-record wait lock	+100	Locks only one record at a time. If the record is already locked, the MicroKernel retries the operation.
Single-record no-wait lock	+200	Locks only one record at a time. If the record is already locked, the MicroKernel returns an error status code.
Multiple-record wait lock	+300	Locks several records concurrently in the same file. If the record is already locked, the MicroKernel retries the operation.
Multiple-record no-wait lock	+400	Locks several records concurrently in the same file. If the record is already locked, the MicroKernel returns an error status code.
No-wait page lock	+500	In a concurrent transaction, tells the MicroKernel not to wait if the page to be changed has already been changed by another active concurrent transaction. This bias can be combined with any of the record locking biases (+100, +200, +300, or +400).

Table 1-6   Data Manipulation Operations

Operation	Code	Description
Insert	2	Inserts a new record into a file.
Update	3	Updates the current record.
Delete	4	Removes the current record from the file.
Insert Extended	40	Inserts one or more records into a file.
Update Chunk	53	Updates specified portions (chunks) of the current record. This operation can also append data to a record or truncate a record.

Sequence of Events in Performing a Btrieve Operation

➤ **To perform a Btrieve operation, your application must complete the following tasks:**

1. Satisfy any prerequisites the operation requires. For example, before your application can perform any file I/O operations, it must make the file available by performing an Open operation (0) on that file.

2. Initialize the parameters that the Btrieve operation requires. The parameters are program variables or data structures that correspond in type and size to the particular values that the MicroKernel expects for an operation.

   For future compatibility, initialize all parameters, whether or not they are used. For parameters of type INTEGER, set the value to binary 0. For character arrays, pass a pointer to a buffer. Initialize the first byte of the buffer to binary 0.

3. Call the appropriate Btrieve function. (Refer to "Btrieve Functions".)

4. Evaluate the results of the function call. Every Btrieve operation returns a status code. Your application must check the status code and take the appropriate action. The operation also returns data or other information to the individual parameters based on the purpose of the operation.

Btrieve Operations

This chapter describes the operations your application can perform using the Btrieve API. For each operation, this chapter presents the following information:

◆ Name, code, and description of the operation.

◆ Parameters—a table indicating which of the six parameter values the operation expects from and returns to your application. A "Sent" parameter is sent from the application to the operation; a "Returned" parameter is returned from the operation to the application when the operation is complete.

◆ Prerequisites—the conditions your application must satisfy for the operation to be successful.

◆ Procedure—the steps for initializing the parameters that the operation requires.

◆ Details—additional information about the operation.

◆ Result—the results of both a successful and an unsuccessful operation. Each operation returns a status code, informing your application of the outcome of the operation. Status Code 0 indicates the operation was successful. A nonzero status code usually indicates a failure. However, some nonzero status codes are informative and appear even when the associated operation is successful—for example, Status Code 60 means the specified reject count has been reached.

◆ Positioning—the effect the operation has on the logical and/or physical currency of the records in a file.

This chapter includes the following Btrieve operations, organized alphabetically:

- ◆ [Abort Transaction (21)](#)
- ◆ [Begin Transaction (19 or 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 (8)](#)
- ◆ [Get Key (+50)](#)
- ◆ [Get Last (13)](#)
- ◆ [Get Less Than (10)](#)
- ◆ [Get Next (6)](#)
- ◆ [Get Next Extended (36)](#)

- ◆ Get Position (22)
- ◆ Get Previous (7)
- ◆ Get Previous Extended (37)
- ◆ Insert (2)
- ◆ Insert Extended (40)
- ◆ 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 Extended (38)
- ◆ Step Previous (35)
- ◆ Step Previous Extended (39)
- ◆ Stop (25)
- ◆ Unlock (27)
- ◆ Update (3)
- ◆ Update Chunk (53)
- ◆ Version (26)

Abort Transaction (21)

The Abort Transaction operation terminates the current transaction and removes the results of all operations performed since the beginning of the transaction. It also unlocks all files and records locked by the transaction.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X					
Returned						

Prerequisites

You must issue a successful Begin Transaction operation (19 or 1019) before you issue an Abort Transaction operation.

Procedure

Set the Operation Code to 21. The MicroKernel ignores all other parameters on an Abort Transaction call.

Result

If the Abort Transaction operation is successful, the MicroKernel returns Status Code 0. The results of all Insert, Update, and Delete operations issued since the beginning of the transaction are removed from the files.

If the Abort Transaction operation is unsuccessful, the MicroKernel returns one of the following status codes:

36   The application encountered a transaction error.

39   A Begin Transaction operation must precede an End/Abort Transaction operation.

Positioning

The Abort Transaction operation has no effect on any file currency information.

Begin Transaction (19 or 1019)

The Begin Transaction operation defines the start of a transaction. Transactions are useful when you need to perform multiple Btrieve operations as a single event. For example, use a transaction if your database would become logically inconsistent if some operations were successful, but at least one operation failed to complete successfully.

By enclosing a set of operations between Begin and End Transaction operations, you can ensure that the MicroKernel does not permanently complete any operations in the set unless you request the completion with an explicit End Transaction operation (20).

The MicroKernel prohibits certain operations during transactions because they have too great an effect on the file or on performance. These operations include Set Owner, Clear Owner, Create Index, Drop Index, and Close (for files updated during the transaction).

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X					
Returned						

Prerequisites

Your application must end or abort any previous transaction before issuing a Begin Transaction operation.

Procedure

Set the Operation Code to 19 to begin an exclusive transaction, or 1019 to begin a concurrent transaction. The MicroKernel ignores all parameters except the Operation Code on any Begin Transaction call.

On any Begin Transaction operation, you can specify default lock biases:

◆ +100—Single wait record lock.

◆ +200—Single no-wait record lock.

◆ +300—Multiple wait record lock.

◆ +400—Multiple no-wait record lock.

On a Begin Concurrent Transaction operation, you can add +500 to the Operation Code (1519), which forces the MicroKernel not to retry the Insert, Update, and Delete operations within a transaction.

In addition, you can combine the +500 bias with a default lock bias (+100, +200, +300, or +400). For example, using 1019 + 500 + 200 (1719) begins a concurrent transaction, suppresses retries for Insert, Update, and Delete operations, *and* specifies single no-wait read locks at the same time.

For more information about transactions and locking, refer to the *Btrieve Programmer's Guide*.

Result

If the Begin Transaction operation is successful, the MicroKernel returns Status Code 0.

If the Begin Transaction operation is unsuccessful, the MicroKernel returns one of the following status codes:

36   The application encountered a transaction error.

37   Another transaction is active.

Positioning

The Begin Transaction operation has no effect on any file currency information.

Clear Owner (30)

The Clear Owner operation removes an owner name that you have previously assigned
to a file with the Set Owner operation. If the file was previously encrypted, the MicroKernel
decrypts the file during a Clear Owner operation.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X				
Returned		X				

Prerequisites

- ◆ The file must be open.
- ◆ No transactions can be active.

Procedure

1. Set the Operation Code to 30.
2. Pass the Position Block that identifies the file to clear.

Result

After a Clear Owner operation, the MicroKernel no longer requires the owner name to
open or modify a file. If you encrypted the data in the file when you assigned the owner,
the MicroKernel decrypts the data during a Clear Owner operation. The more data that
was encrypted, the longer the Clear Owner operation takes.

If the Clear Owner operation is unsuccessful, the MicroKernel returns one of the following status codes:

   3   The file is not open.

  41   The MicroKernel does not allow the attempted operation.

Positioning

The Clear Owner operation has no effect on any file currency information.

Close (1)

The Close operation closes the file associated with a specified Position Block and releases any locks your application has executed for the file. Your application should always perform a Close operation when it has finished accessing a file. After a Close operation, your application cannot access the file again until it issues another Open operation (0) for that file.

You can close a file even while inside a transaction. However, the Close operation does not end the transaction; you must explicitly end or abort the transaction. If you abort the transaction, changes made inside the transaction are aborted; if you end the transaction, changes are committed.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X				
Returned						

Prerequisites

◆ The file must be open.

Procedure

1. Set the Operation Code to 1.

2. Pass a valid Position Block for the file to close.

Result

If the Close operation is successful, the Position Block for the closed file is no longer valid. Your application can use it for another file or use the data area for other purposes.

If the Close operation fails, the file remains open and the MicroKernel returns the following status code:

3   The file is not open.

Positioning

The Close operation destroys both the physical and the logical currency information of the file.

Continuous Operation (42)

The Continuous Operation operation allows you to perform system backups without closing active Btrieve files. Any changes you make while a file is being backed up are stored in a temporary file called a delta file. Except for changes written to the delta file, the system backup includes the contents of all files placed in continuous operation mode. The MicroKernel automatically rolls the delta file changes into the backed up files when those files are taken out of continuous operation mode.

This operation also allows you to copy a file while that file is still active.

> **Note**
>
> This operation is available only to applications running local on a server machine.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X		X	X		X
Returned			X	X		

> **Note**
>
> Values for the Data Buffer parameter and the Data Buffer Length parameter are required only if the value of the Key Number parameter is 0 (which starts continuous operation mode) or 2 (which ends continuous operation mode). The following sections discuss these Key Number values. A Data Buffer Length of zero is required for a Key Number parameter of 1.

Procedure

➤ **To start continuous operation mode, perform the following steps:**

  1. Define a file or a set of files for backup, or add a file to the set of files currently defined for backup.

   a. Set the Operation Code to 42.

   b. Place the names of the files you want to place in continuous operation mode into the Data Buffer parameter. Include the full pathname, excluding only the server name. Separate the names with commas and terminate the list of names with a binary 0.

   The following example is for Windows NT servers:

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

   The following example is for NetWare servers:

   ```
   sys:\acct\march.mkd,sys:\acct\april.mkd
   ```

   c. Place the length of the name (or names) in the Data Buffer Length parameter. This value must be equal to or greater than the actual length of the names (including binary zeros) in the Data Buffer itself. For example, the preceding names require a Data Buffer Length of 40 or greater.

   d. Set the Key Number parameter to 0.

  2. Perform the backup.

  3. End continuous operation mode.

   a. Set the Operation Code to 42.

   b. Set the Key Number parameter to 1.

   To end continuous operation on one or more specific files, set the Key Number parameter to 2, and then place the filenames in the Data Buffer parameter as described in step 1b. Also, place the length of the name (or names) in the Data Buffer Length parameter as described in step 1c.

Details

When defining the set of files to be backed up, keep in mind the following information:

- ◆ The MicroKernel does not consider the absence of filenames in the Data Buffer to be an error. If it finds no filenames, the MicroKernel takes no action on the Continuous Operation operation.

- ◆ The presence of duplicate filenames in the Data Buffer does not affect how the Continuous Operation operation works. The MKDE places the specified file in continuous operation mode only once.

- ◆ No two files are allowed to share the same filename and differ only in their extension. This is because the name of the delta file uses the corresponding file's name with ".^^^" for the extension.

- ◆ An application can iteratively call the Continuous Operation operation to add more names to the list of files to be placed in continuous operation mode. However, this action can corrupt a backup when referential integrity (RI) constraints are placed on any of the files by Scalable SQL.

  The MicroKernel returns Status Code 88 if a file is specified that is already in continuous operation mode.

When writing a server-based application that calls the Continuous Operation operation, make sure you call btrvID, and use a valid client ID so you can begin and end continuous operation under the same client.

Btrieve allows you to define multiple backup sets by specifying a different client ID for each backup set through the btrvID function. However, two sets cannot contain the same files.

While the MicroKernel rolls changes from the delta file into the data file, users can continue to update, insert, and read the Btrieve file just as they normally would. The

MicroKernel appends new pages to the delta file while rolling in changes, if an insert requires such an action. No changes are lost.

Note

Never delete a delta file manually.

If your application uses the btrv function, do not unload the application while any file is in continuous operation mode. If you do, you may be unable to remove the affected files from continuous operation mode. This is because the default client ID that the MicroKernel originally assigned as the owner of the affected files may have been reassigned to another application. Because the MicroKernel no longer knows the proper owner of the affected files, it is unable to remove those files from continuous operation mode.

If the system crashes while in continuous operation mode or while the MicroKernel is rolling the changes from a delta file into the file, then the MicroKernel rolls all changes into the file when it is first opened after the system is rebooted.

Result

If the Continuous Operation operation is successful, the MicroKernel returns Status Code 0, but it returns no values either in the Data Buffer or in the Data Buffer Length parameter.

If the operation is unsuccessful, the MicroKernel returns one of the following status codes:

11   The specified filename is invalid.

12   The MicroKernel cannot find the specified file.

41   The MicroKernel does not allow the attempted operation.

51   The owner name is invalid.

88   The application encountered an incompatible mode error.

91   The application encountered a server error.

In addition to the preceding codes, your application can return standard I/O error codes such as Status Code 18.

If the MicroKernel returns a nonzero status code, the Continuous Operation operation returns in the Data Buffer the portion of the input string that generated the error. If no input string was used, the Data Buffer contains the file names that caused the error. The Data Buffer Length reflects the length of the output string in the Data Buffer. At this point, the Data Buffer Length contains the length of that filename.

Positioning

The Continuous Operation operation does not establish any currency on the file.

Create (14)

The Create operation lets you generate a file from within your application.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X		X	X	X	X
Returned						

Prerequisites

If you are creating an empty file over an existing file, ensure that the existing file is closed before executing the Create operation.

Procedure

1. Set the Operation Code to 14.

2. Specify the file specifications, key specifications, and any alternate collating sequences in the Data Buffer as described in "Details". (All the values for the file specifications and key specifications you store in the Data Buffer must be in binary format.)

3. Specify the Data Buffer Length. This is the length of the buffer that contains the Create specifications, not the length of the records in the file.

4. Specify the pathname for the file in the Key Buffer. Be sure to terminate the pathname with a blank or binary zero. The pathname can be up to 80 characters long, including the volume name and the terminator.

For more information about path names Btrieve supports, refer to *Getting Started*.

Note

For your application to be compatible with Scalable SQL, limit the pathname to 64 characters.

5. If you want the MicroKernel to warn you that a file of the same name already exists, set the Key Number parameter to –1. Otherwise, set the Key Number parameter to 0.

Details

illustrates the order in which the file and key specifications must be stored.

Table 2-1   Data Buffer Structure for Create Operation

Element	Description	Unsigned Data Type[1]	Length (in Bytes)
File Specification	Logical Record Length	short int	2
	Page Size	short int	2
	Number of Indexes	short int	2
	Reserved	char	4
	File Flags	short int	2
	Number of Duplicate Pointers To Reserve	char	1
	Not Used	char	1
	Allocation	short int	2
Key Specification (repeated for each key segment)	Key Position	short int	2
	Key Length	short int	2
	Key Flags	short int	2
	Reserved	char	4
	Extended Data Type	char	1
	Null Value	char	1
	Not Used	char	2
	Manually Assigned Key Number	char	1
	ACS Number	char	1
ACS Number 0	ACS	char	265
...	...	char	265

Table 2-1   Data Buffer Structure for Create Operation *continued*

Element	Description	Unsigned Data Type[1]	Length (in Bytes)
ACS Number *x*	ACS	char	265

1 For simplification, this table shows C language data types only. <u>Table A-2</u> lists the data types in other languages.

Note

You must allocate the "not used" and "reserved" areas of the Data Buffer, even though the MicroKernel does not use them for a Create operation. Initialize the reserved areas to zero to maintain compatibility with future releases.

File Specification

Store the file specification in the first 16 bytes of the Data Buffer. The bytes are numbered beginning with 0. Store the information for the record length, page size, and number of indexes as integers.

Logical Record Length. (Offset 0x00) The logical record length is the number of bytes of fixed-length data in the file. (Do not include variable-length data in the logical record length.)

Page Size. (Offset 0x02) You can specify any multiple of 512 up to 4,096. For more information about choosing the optimal page size, refer to the *Btrieve Programmer's Guide*.

Number of Indexes. (Offset 0x04) The number of indexes is the number of keys (not key segments) you are defining for the file. To create a data-only file, set the number of indexes to 0.

File Flags. (Offset 0x0A) The bit settings in the File Flags word specify file attributes. Table 2-2 shows the binary, hexadecimal, and decimal representations of the file flag values.

Table 2-2   File Flag Values

Attribute	Constant	Binary	Hex	Decimal
Variable Length Records	VAR_RECS	0000 0000 0000 0001	1	1
Blank Truncation	BLANK_TRUNC	0000 0000 0000 0010	2	2
Page Preallocation	PRE_ALLOC	0000 0000 0000 0100	04	4
Data Compression	DATA_COMP	0000 0000 0000 1000	08	8
Key-Only File	KEY_ONLY	0000 0000 0001 0000	10	16
Balanced Index	BALANCED_KEYS	0000 0000 0010 0000	20	32
10% Free Space	FREE_10	0000 0000 0100 0000	40	64
20% Free Space	FREE_20	0000 0000 1000 0000	80	128
30% Free Space	FREE_30	0000 0000 1100 0000	C0	192
Reserve Duplicate Pointers	DUP_PTRS	0000 0001 0000 0000	100	256
Include System Data[1]	INCLUDE_SYSTEM_DATA	0000 0010 0000 0000	200	512
Do Not Include System Data*	NO_INCLUDE_SYSTEM_DATA	0001 0010 0000 0000	1200	4,608
Key Number	SPECIFY_KEY_NUMS	0000 0100 0000 0000	400	1,024
Use VATs	VATS_SUPPORT	0000 1000 0000 0000	800	2,048

1 If you do not explicitly specify whether to include system data in the file, Btrieve uses the current setting of the MicroKernel configuration option "Include System Data."

Avoid using incompatible flags; flags are incompatible if they use the same bit positions. Unused bits are reserved for future use. Set them to 0.

To combine file attributes, add their respective File Flag values. For example, to specify a file that allows variable-length records and uses blank truncation, initialize the File Flags word to 3 (2 + 1). The MicroKernel ignores the Blank Truncation and Free Space bits if the Variable Length Records bit is set to 0.

If you set the Page Preallocation bit, use the last 2 bytes in the file specification block (allocation) to store an integer value that specifies the number of pages to preallocate to the file. If you set the Data Compression bit, the MicroKernel ignores the Variable Length Records bit. If you set the Key-only File bit, the MicroKernel ignores the System Data bits.

> **Note**
>
> The MicroKernel automatically uses data compression on files that use system data and have a record length greater than 4,076 bytes.

Set the Duplicate Pointers bit if you anticipate adding an index sometime after creating a file, and if that index has many duplicate values. Setting this bit causes the MicroKernel to reserve space in each record of the file for pointers that link the duplicate values. By reserving this space, you can possibly lower retrieval time and save disk space, especially if the keys are long and you anticipate that many records will have duplicate key values.

> **Note**
>
> You can reserve duplicate pointer space only for indexes that are added after file creation. Therefore, when reserving space for pointers to duplicate values, do *not* include space for the indexes created during this Create operation. Also, the MicroKernel does not reserve duplicate pointer space for any key you specify as a repeating-duplicatable key.

Set the Key Number bit if you need to assign a specific number to a key, and place the desired key number in the Manually Assigned Key Number element (offset 0x0E) of the Key Specification block. The MicroKernel does not require consecutive key numbers;

files can have gaps between key numbers. When a key is created, by default the MicroKernel assigns the lowest available key number to that index (beginning with 0). However, some applications may require a key number different from the default assignment.

Set the Use VATs bit if the file uses Variable-tail Allocation Tables. To use VATs, a file must use variable-length records.

Number of Duplicate Pointers to Reserve. (Offset 0x0C) You can specify the number of duplicate pointers to reserve for each key. Use this element only if you specified the Reserve Duplicate Pointers file flag. For more information about duplicatable keys, refer to the *Btrieve Programmer's Guide*.

Allocation. (Offset 0x0E) You can specify the number of pages to preallocate. Use this element only if you specified the Page Preallocation file flag. For more information about page preallocation, refer to the *Btrieve Programmer's Guide*.

Key Specification Blocks

Place the key specification blocks immediately after the file specification. Allocate a 16-byte key specification block for each key segment in the file. Store the information for the Key Position and the Key Length as integers.

The maximum number of key segments allowed depends on the file's page size. The following table shows these values:

Page Size	Maximum Key Segments
512	8
1024	23
1536	24
2048, 2560, 3072, or 3584	54
4096	119

Key Position. (Offset 0x00) The key position is the byte offset at which the key or key segment begins.

Key Length. (Offset 0x02) The length of the key or key segment. The maximum length of a key, including all key segments, is 255 bytes.

Key Flags. (Offset 0x04) The bit settings in the Key Flags word specify key attributes. Table 2-3 shows the binary, hexadecimal, and decimal representations of the Key Flags values.

Table 2-3   Key Flag Values

Attribute	Constant	Binary	Hex	Decimal
Linked Duplicates	DUP	0000 0000 0000 0001	1	1
Modifiable Key Values	MOD	0000 0000 0000 010	2	2
Use Old Style BINARY Data Type	BIN	0000 0000 0000 0100	4	4
Use Old Style STRING Data Type[1]		0000 0000 0000 0000	0	0
Null Key (All Segments)	NUL	0000 0000 0000 1000	8	8
Segmented Key	SEG	0000 0000 0001 0000	10	16
Use Default ACS	ALT	0000 0000 0010 0000	20	32
Use Numbered ACS in File	NUMBERED_ACS	0000 0100 0010 0000	420	1,056
Use Named ACS	NAMED_ACS	0000 1100 0010 0000	C20	3,104
Descending Sort Order	DESC_KEY	0000 0000 0100 0000	40	64
Repeating Duplicates	REPEAT_DUPS_KEY	0000 0000 1000 0000	80	128
Use Extended Data Type	EXTTYPE_KEY	0000 0001 0000 0000	100	256
Null Key (Any Segment)	MANUAL_KEY	0000 0010 0000 0000	200	512
Case Insensitive Key	NOCASE_KEY	0000 0100 0000 0000	400	1,024

1 Bit 2 and Bit 8 must be 0.

Avoid using incompatible flags; flags are incompatible if they use the same bit positions. Unused bits are reserved for future use. Set them to 0.

To combine key attributes, add their respective Key Flags values. For example, if the key is an extended type, part of a segmented key, and to be collated in descending order, initialize the Key Flags word to 336 (256 + 16 + 64).

The Segmented Key attribute indicates that the next key specification block in the Data Buffer refers to the next segment of the same key. Follow these rules for segmented keys:

- ◆ All segments of the same key must have the same Duplicate Key Values, Repeating Duplicates, Modifiable Key Values, and Null Key values. (If you specify the Null Key attribute, either All Segments or Any Segment, you can assign different null values for individual segments.)

- ◆ All segments of the same key must use the same ACS.

- ◆ Individual segments of the same key can have different Descending Sort Order and Extended Data Type values.

ACSs are applicable only to STRING, LSTRING, and ZSTRING keys. You cannot define a key that is both case-insensitive and uses an ACS. In a file in which a key has an ACS designated for some segments but not for others, the segments that designate an ACS are sorted by the specified ACS; the segments with no ACSs are sorted according to their respective types.

Extended Data Type. (Offset 0x0A) You specify the Extended Data Type in byte 10 of the Key Specification block as a binary value. Table 2-4 shows the codes for the extended data types.

Table 2-4   Extended Data Types

Type	Code	Type	Code
STRING	0	BFLOAT	9
INTEGER	1	LSTRING	10
FLOAT	2	ZSTRING	11
DATE	3	UNSIGNED BINARY	14
TIME	4	AUTOINCREMENT	15
DECIMAL	5	NUMERICSTS	17
MONEY	6	NUMERICSA	18
LOGICAL	7	CURRENCY	19
NUMERIC	8	TIMESTAMP	20

Extended data type codes 12, 13, and 16 are reserved for future use.

You can define the STRING and UNSIGNED BINARY data types as either standard or extended types. This maintains compatibility with applications that were developed with earlier versions of Btrieve, while allowing newer applications to use extended data types exclusively.

Regarding the data type you assign to a key, Btrieve does not ensure that the records you input adhere to the data types defined for the keys. For example, you could define a TIMESTAMP key in a file, but store a character string there. Your Btrieve application would work fine, but an ODBC application that tries to access the same data using the ODBC TIMESTAMP format might fail, because the byte format could be different and the

algorithms used to generate the timestamp value could be different. For complete descriptions of the data types, refer to <u>"Data Types"</u>.

Null Value. (Offset 0x0B) of the Key Specification block represents an exclusion value for the key. If you have defined a key as a null key, you must supply a value for the MicroKernel to recognize as the null value for each key segment.

Manually Assigned Key Number. (Offset 0x0E) The MicroKernel allows an application to assign specific key numbers when creating a file with indexes. To manually assign key numbers to each index for a file, specify each key's number as a binary value in byte 14 of the key specification block, and set the Key Number bit (0x400) in the File Flags word.

Key numbers must be unique to the file and must be specified in ascending order, beginning with key 0. They must also be valid (less than the maximum number of keys for the file's page size).

The ability to manually assign key numbers complements to the ability to delete a key and not have the MicroKernel renumber all keys that have a key number greater than the deleted key. For example, if an application drops an index and instructs the MicroKernel not to renumber higher-numbered keys, and if a user then clones the affected file without assigning specific key numbers, the cloned file has different key numbers than the original.

ACS Number. (Offset 0x0F) For keys that use a specific ACS, offset 0x0F in the key specification block provides the ACS number by which to collate the key. The ACS number is based on its position in the Data Buffer. The first ACS following the last key specification block is ACS number 0. Following ACS 0 is ACS 1, which is followed by ACS 2, and so on.

Alternate Collating Sequence

Your application specifies ACSs one after the other immediately following the last key specification block in the Data Buffer.

User-Defined ACSs. To create an ACS that sorts string values differently from the ASCII standard, your application must place 265 bytes directly into the Data Buffer, using the following format:

Table 2-5   Data Buffer for Creating a User-Defined ACS

Offset	Length (in Bytes)	Description
0	1	Signature byte. Specify 0xAC.
1	8	A unique 8-byte name that identifies the ACS to the MicroKernel.
9	256	A 256-byte map. Each 1-byte position in the map corresponds to the code point having the same value as the position's offset in the map. The value of the byte at that position is the collating weight assigned to the code point. For example, to force code point 0x61 ('a') to sort with the same weight as code point 0x41 ('A'), place the same values at offsets 0x61 and 0x41.

For examples of user-defined ACSs, refer to the *Btrieve Programmer's Guide.*

Locale-Specific ACSs. To specify an ACS that sorts string values according to a locale-specific collating sequence, your application must place 265 bytes directly into the Data Buffer, using the following format:

Table 2-6   Data Buffer for Specifying a Locale-Specific ACS

Offset	Length (in Bytes)	Description
0	1	Signature byte. Specify 0xAD.
1	2	Country ID (Intel format). See your operating system's documentation for more information about national language support.
3	2	Code page ID (Intel format). See your operating system's documentation for more information about national language support.
5	260	Filler.

To use the default locale-specific ACS, specify a value of 0xFFFF for the Country ID and the Code page ID.

> **Note**
>
> Windows NT clients and servers do not support passing a locale's country ID and code page number to Btrieve. If you want to use an ACS other than the one for the default locale, you must create a user-defined ACS or specify an international sort rule (ISR).

International Sort Rules (ISRs). To specify an ISR, your application must place 265 bytes directly into the Data Buffer, using the following format:

Table 2-7   Data Buffer for Specifying an ISR ACS

Offset	Length (in Bytes)	Description
0	1	Signature byte. Specify 0xAE.
1	16	A unique 16-byte name that identifies the ISR table to the MicroKernel. Refer to the *Btrieve Programmer's Guide* for a list of ISR table names.
17	248	Filler.

Data Buffer Length

The Data Buffer Length must be long enough to include the file specifications, the key specifications, and any ACSs that have been defined. Do not specify the file's record length in this parameter.

For example, to create a file that has two keys of one segment each and an ACS, the Data Buffer for the Create operation should be at least 313 bytes long, as follows:

File Spec	+	Key1 Spec	+	Key2 Spec	+	ACS	
16	+	16	+	16	+	265	= 313

Key Number

The Create operation's Key Number parameter is used to determine if the MicroKernel warns you when a file of the same name already exists.

Result

If the Create operation is successful, the MicroKernel either warns you of the existence of a file with the same name or creates the new file according to your specifications. The new file does not contain any records. The Create operation does not open the file. Your application must perform an Open operation before it can access the file.

If the Create operation is unsuccessful, the MicroKernel returns one of the following status codes:

2	The application encountered an I/O error.
11	The specified file name is invalid.
18	The disk is full.
22	The data buffer length is too short.
24	The page size or data buffer size is invalid.
25	The application cannot create the specified file.
26	The number of keys specified is invalid.
27	The key position is invalid.
28	The record length is invalid.
29	The key length is invalid.
48	The alternate collating sequence definition is invalid.
49	The extended data type is invalid.
59	The specified file already exists.
104	The MicroKernel does not recognize the locale.
105	The file cannot be created with Variable-tail Allocation Tables (VATs).

134   The MicroKernel cannot read the International Sorting Rule.

135   The specified International Sort Rule table is corrupt or otherwise invalid.

Positioning

The Create operation establishes no currency on the file.

Create Index (31)

The Create Index operation adds a key to an existing file.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X		X
Returned						

Prerequisites

◆ The file must be open.

◆ The number of existing key segments in the file must be less than or equal to maximum number of key segments allowed minus the number of key segments to be added.

The maximum number of key segments allowed depends on the file's page size. The following table shows these values:

Page Size	Maximum Key Segments
512	8
1024	23
1536	24
2048, 2560, 3072, or 3584	54
4096	119

◆ Ensure that the key flags, position, and length of the new key are appropriate for the file to which you are adding the key.

◆ No transactions can be active.

Procedure

1. Set the Operation Code to 31.

2. Pass the Position Block for the file to which to add the key.

3. For each segment in the key, store a 16-byte key specification block in the Data Buffer. Use the same structure as defined in Table 2-1. Store the information for the key position and the key length as integers. If you are rebuilding the system-defined log key (also called system data), the Data Buffer must be at least 16 bytes long and initialized to zeroes.

4. To define an ACS for the new key, perform one of the following steps:

   - To use the default ACS, which is the first ACS already defined in the file, specify the Use Default ACS attribute in the Key Flags word.

   - To define a new ACS, specify the Use Numbered ACS attribute in the Key Flags word and set the ACS Number field to zero (0). In addition, store the 265-byte ACS after the last key specification block in the Data Buffer.

   - To specify an existing ACS by name, specify the Use Named ACS attribute in the Key Flags word and set the ACS Number field to zero (0). In addition, store the name of the ACS at the beginning of the 265-byte block after the last key specification block in the Data Buffer. (The remainder of the ACS block after the name is ignored.) The name must be in one of the following formats:

Table 2-8   ACS Name Formats

ACS Type	Length (in Bytes)	Description
User-defined ACS	1	Signature 0xAC
	8	ACS table name
Locale-specific ACS	1	Signature 0xAD
	2	Country ID
	2	Code page ID
ISR	1	Signature 0xAE
	16	ISR table name

5. Set the Data Buffer Length parameter to the number of bytes in the Data Buffer. For a new key with no ACS (or one that uses the default ACS), use the following formula to determine the correct Data Buffer Length:

16 * (# of segments)

If the new key specifies an ACS other than the default, use the following formula to determine the correct Data Buffer Length:

16 * (# of segments) + 265

6. To assign a specific Key Number to the key being created, add the desired key number to 0x80, and place the sum in the Key Number parameter. If you are rebuilding the system-defined log key (also called system data), specify 0xFD (that is, key number 125 plus 128).

Details

The MicroKernel allows you to assign specific key numbers when creating a key. This capability complements the ability to delete a key and not have the MicroKernel renumber all keys that have a key number greater than that of the deleted key. If an application drops an index and instructs the MicroKernel not to renumber higher-numbered keys, and a user then clones the affected file *without* assigning specific key numbers, the cloned file has different key numbers than the original.

If you define an ACS in the Data Buffer, the MicroKernel first checks for an existing ACS (using the name you specified) before adding it to the file. If the MicroKernel finds an existing ACS with the name you specified, the MicroKernel does not duplicate the ACS definition in the file, but does associate the ACS with the new key.

If you specify the Use Named ACS attribute in the Key Flags word, the MicroKernel uses the ACS name supplied in the Data Buffer to locate an ACS of the same name within the file, then assigns that ACS to the new key.

If a file is opened by more than one MicroKernel and a client initiates a Create Index process, remote clients can perform Get and Step operations on the same file while the MicroKernel creates the key.

If the key being created is not an AUTOINCREMENT key, the Get and Step operations of remote clients can have lock biases, and when the Create Index process is completed, you can update and delete the locked records without issuing additional read operations. This is possible because the MicroKernel does not have to change the images of the records in order to create the key.

However, if the key being created is an AUTOINCREMENT key, the MicroKernel has to both build the index and change every record with a zero value in the appropriate field. Remote clients that perform Get or Step operations without a lock bias before or during the key creation can receive Status Code 80 when they execute an update or delete operation after the successful completion of the key creation.

Also, the MicroKernel returns Status Code 84 if a client attempts to create an AUTOINCREMENT key while another client has locked a record. Similarly, the MicroKernel returns Status Code 85 if a client attempts to execute a Get or Step operation with a lock bias during index creation for an AUTOINCREMENT key by another client.

Result

The MicroKernel immediately adds the new key to the file. The time required for this operation depends on the total number of records to be indexed, the size of the file, and the length of the new index.

If the Create Index operation is successful, the number of the new key is either the number you specified or one of the following:

◆ For files that have no gaps between key numbers, the key number is one higher than the previous highest key number.

◆ For files that have gaps between key numbers, the key number is the lowest missing key number.

You can use the new key to access your data as soon as the operation completes.

If the Create Index operation is unsuccessful, the MicroKernel drops whatever portion of the new index it has already built. The file pages allocated to the new index prior to the error are placed on a list of free space for the file and reused when you insert records or create another key.

If the operation fails during the creation of an AUTOINCREMENT key, any values that have already been altered remain altered. The MicroKernel can return the following status codes:

22	The data buffer length is too short.
27	The key position is invalid.
41	The MicroKernel does not allow the attempted operation.
45	The specified key flags are invalid.
49	The extended data type is invalid.
56	An index is incomplete.
84	The record or page is locked.
85	The file is locked.
104	The MicroKernel does not recognize the locale.
134	The MicroKernel cannot read the International Sorting Rule.
135	The specified International Sort Rule table is corrupt or otherwise invalid.
136	The MicroKernel cannot find the specified Alternate Collating Sequence in the file.

If processing is interrupted during the creation of a key, you can access the data in the file through the file's other keys. However, the MicroKernel returns a nonzero status code if you try to access data by the incomplete index. To correct this problem, drop the incomplete index with a Drop Index operation (32) and reissue the Create Index operation.

Positioning

The Create Index operation has no effect on any file currency information.

Delete (4)

The Delete operation removes an existing record from a file. The space that the deleted record occupied is then available for inserting new records.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X				
Returned		X				

Prerequisites

- ◆ The file must be open.
- ◆ You have established physical or logical currency in the file. Operations that satisfy this requirement are: Get (except extended Gets or Get Key), Step (except extended Steps), Insert, and Update.
- ◆ If you want to delete a record while inside a transaction, the record must have been retrieved while inside the transaction.

Procedure

1. Set the Operation Code to 4.
2. Pass the Position Block of the file that contains the record to be deleted.

Details

The Delete operation is not a valid operation if performed immediately after an extended Get or extended Step operation.

When performing a Delete operation immediately following a Get operation, do *not* change the Key Number that the Get operation returns. If you do, the MicroKernel deletes the record successfully; however, it returns Status Code 7 on the first Get operation performed after the deletion.

The MicroKernel does not allow Delete operations after a Get Key operation (+50). Before the MicroKernel performs a Delete operation, it compares the current usage count of the data page it intends to modify with the usage count of the data page when the record was read. To obtain the usage count, the MicroKernel must read the data page.

Because the Get Key operation does not read the data page, no usage count is available for comparison on the Delete. The Delete fails because the MicroKernel cannot perform its passive concurrency conflict checking without the compare. When the Delete fails, the MicroKernel returns Status Code 8.

Result

If the Delete operation is successful, the MicroKernel removes the record from the file, releases the record lock (if one existed for the deleted record), and adjusts all key indexes to reflect the deletion.

If the Delete operation is unsuccessful, the MicroKernel returns one of the following status codes:

8	The current positioning is invalid.
80	The MicroKernel encountered a record-level conflict.
83	The application encountered an incompatible mode error.

Positioning

The Delete operation destroys the complete physical location information and the logical current record position but leaves the positions of the logical next record and logical previous record unchanged.

Drop Index (32)

The Drop Index operation deletes a key from an existing file.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X				X
Returned						

Prerequisites

◆ The file must be open.

◆ The key must exist in the file.

◆ No transactions can be active.

Procedure

1. Set the Operation Code to 32.

2. Pass the Position Block of the file that contains the key to drop.

3. Store the number of the key to drop in the Key Number parameter. To drop the system-defined log key (also called system data), specify 125.

Details

You may want to drop the system-defined log key (also called system data) and rebuild it if it becomes corrupted. After dropping the system-defined log key, rebuild it using the Create Index (31) operation.

When you delete a key, the MicroKernel automatically renumbers all higher-numbered keys, unless you specify otherwise. The MicroKernel renumbers by decrementing the higher-numbered keys by 1. For example, suppose you have a file with key numbers 1, 4, and 7. If you drop key 4, the MicroKernel renumbers the keys as 1 and 6.

If you do not want the MicroKernel to automatically renumber keys, add a bias of 128 to the value you supply for the Key Number parameter. This allows you to leave gaps in the key numbering; consequently, you can drop a damaged index and then rebuild it without affecting the numbering of other keys in the file. You rebuild the index using the Create Index operation (31), which allows you to specify a key number.

However, if you delete a key and do not renumber higher-numbered keys and a user then clones the affected file without assigning specific key numbers, the cloned file has different key numbers than the original. (Users can clone files using the Btrieve Maintenance utility. Cloning is the process of creating a new, empty file with the same statistics as an existing file.)

Result

If the Drop Index operation is successful, the MicroKernel places the pages that were allocated to that index on a list of free space for later use. Unless you specify otherwise, the MicroKernel renumbers the higher-numbered keys.

If the Drop Index operation is unsuccessful, the MicroKernel returns one of the following status codes:

6	The key number parameter is invalid.
41	The MicroKernel does not allow the attempted operation.

If processing is interrupted while the MicroKernel is dropping an index, you can access the data in the file by the file's other keys. The MicroKernel returns Status Code 56 if you

try to access the file by an incomplete index. If processing is interrupted, reissue the Drop Index operation.

Positioning

The Drop Index operation has no effect on physical file currency information. However, dropping the key used to establish the last logical currency destroys the logical currency.

End Transaction (20)

The End Transaction operation completes a transaction and makes the appropriate changes to the data files. It also unlocks all files and records locked by the transaction.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X					
Returned						

Prerequisites

Before issuing an End Transaction operation, your application must issue a successful Begin Transaction operation (19 or 1019).

Procedure

Set the Operation Code to 20. While the MicroKernel ignores all other parameters on an End Transaction call, you should initialize them to 0 to ensure compatibility with future releases.

Result

If the End Transaction operation is successful, all the operations within the transaction are recorded in your file. Your application cannot abort a transaction after an End Transaction operation.

If the End Transaction operation is unsuccessful, the MicroKernel returns the following status code:

    38   The MicroKernel encountered a transaction control file I/O error.

Positioning

The End Transaction operation has no effect on any file currency information.

Find Percentage (45)

The Find Percentage operation is one of two Btrieve operations that window-oriented applications can use to implement scroll bars. The other is the Get By Percentage operation (44). Find Percentage finds the position of a record either relative to a key path or as the record's physical location within the file. The position is expressed as a percentage value. See the "Result" section for a definition of the range of percentage values.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X	X	X
Returned		X	X	X		

Note

When seeking the percentage relative to a key path, Find Percentage does not require an input value for the Data Buffer parameter. When seeking the percentage as relative to a record's physical location within the file, Find Percentage does not require an input value for the Key Buffer parameter.

Prerequisites

- ◆ The file must be open.

- ◆ If you are seeking the percentage relative to a key path, the file cannot be a data-only file.

- ◆ When you are seeking the percentage by a record's physical location in the file, you must provide the 4-byte physical location of the record. You can retrieve this location with a Get Position operation (22).

Procedure

1. Set the Operation Code to 45.

2. Pass the Position Block for the file.

3. If you are seeking the percentage relative to the record's physical location within the file, store the record's physical address in the Data Buffer parameter. Otherwise, you do not need to provide a value for the Data Buffer parameter.

4. Set the Data Buffer Length to a minimum of 4 bytes. (This 4-byte minimum is a requirement of the MicroKernel's internal implementation.)

5. If you are seeking the percentage relative to a key path, set the Key Buffer parameter to the key value. Otherwise, you do not need to provide a value for the Key Buffer parameter.

6. Set the Key Number parameter as follows:

    a. If you are seeking the percentage by a key path, set the Key Number parameter to the actual key number.

    b. If you are seeking the percentage by the record's physical location, set the Key Number parameter to –1 (0xFF).

Details

The Find Percentage operation is provided specifically to support scroll bar implementation. Because many factors affect the accuracy of this operation—that is, whether the returned percentage value accurately reflects the position of the record or key value—you should not rely on the accuracy of this operation for other purposes.

To optimize the Find Percentage operation, the MicroKernel assumes that a file has an even distribution of records among the data pages and keys among the index pages. However, distribution can be affected by the following situations:

◆ The file is not index balanced, and a large number of records within the same range of keys has been deleted.

- ◆ A large number of records within the same range of physical addresses has been deleted.

- ◆ The file contains numerous duplicate key values, and the key is a linked-duplicatable key.

Result

If the Find Percentage operation is successful, the MicroKernel returns the position of the specified key value or record to the Data Buffer. This position is expressed as a percentage of the offset into the key path or file and is a value in the range of 0 (0 percent) through 10,000 (100.00 percent).

The percentage value is returned as an integer in low-byte, high-byte order. For example:

Returned Value Hex	Returned Value Dec	Percentage in Key Path or File
88h 13h	00 50	50%

The MicroKernel also returns a Data Buffer Length of 4 if the operation is successful.

If the Find Percentage operation is unsuccessful, the MicroKernel returns one of the following status codes:

- 3 The file is not open.
- 6 The key number parameter is invalid.
- 7 The key number has changed.
- 8 The current positioning is invalid.
- 9 The operation encountered the end-of-file.

22	The data buffer parameter is too short.
41	The MicroKernel does not allow the attempted operation.
43	The specified record address is invalid.
82	The MicroKernel lost positioning.

Positioning

The Find Percentage operation does not change any currency information.

Get By Percentage (44)

The Get By Percentage operation is one of two Btrieve operations that can be used by window-oriented applications for implementing scroll bars. The other is the Find Percentage operation (45). Get By Percentage retrieves a record by that record's position in the file, where the position is based on a percentage value you supply when you call the operation. You must also specify whether the position is relative to a specified key path or represents the record's actual physical location in the file.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X		X
Returned		X	X	X	X	

Note

The Get By Percentage operation, when seeking the record by its physical location in the file, does not return any information in the Key Buffer parameter.

Prerequisites

◆ The file must be open.

◆ If you are seeking the record relative to a key path, the file cannot be a data-only file.

Procedure

1. Set the Operation Code to 44. Optionally, you can include a lock bias:

   - 100—Single wait record lock.

   - 200—Single no-wait record lock.

   - 300—Multiple wait record lock.

   - 400—Multiple no-wait record lock.

   For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Store the percentage value as a 16-bit integer in the Data Buffer. See the "Details" section for the acceptable range of percentage values and related information.

4. Set the Data Buffer Length to a value greater than or equal to the length of the largest possible record that could be returned. (The MicroKernel's internal implementation requires that you set the Data Buffer Length value to a minimum of 4 bytes.)

5. Set the Key Number parameter.

   a. If you are seeking the record by a key path, set the Key Number parameter to the actual key number. To use the system-defined log key (also called system data), specify 125.

   b. If you are seeking the record by the record's physical position in the file, set the Key Number parameter to −1 (0xFF).

Details

The range of acceptable percentage values for the Data Buffer parameter is from 0 (indicating the beginning of the key path or file) through 10,000 (the end of the key path or file). Be sure to store the value as an integer (in low-byte, high-byte order). For

example, to seek to the 50 percent point in the file, use a value of 5,000 (0x1388). After byte-swapping 0x1388, store 0x88 and 0x13 in the first two bytes of the Data Buffer parameter.

The Get By Percentage operation is provided specifically to support scroll bar implementation. Because many factors affect the accuracy of this operation—that is, whether the returned record is positioned at the actual percentage point you specify in the file—you should not rely on the accuracy of this operation for other purposes.

To optimize the Get By Percentage operation, the MicroKernel assumes that a file has an even distribution of records among the data pages and keys among the index pages. However, distribution can be affected by the following situations:

◆ The file is not index balanced, and a large number of records within the same range of keys has been deleted.

◆ A large number of records within the same range of physical addresses has been deleted.

◆ The file contains numerous duplicate key values, and the key is a linked-duplicatable key.

Result

If the Get By Percentage operation is successful, the MicroKernel returns to the Data Buffer a record that is either from the designated position relative to the specified key path or from the physical position in the file. The MicroKernel returns the length of the record in bytes into the Data Buffer Length parameter. If the operation seeks the record by a key path, the MicroKernel returns the key value for the specified key path in the Key Buffer parameter. If the operation seeks the record by physical record order, the MicroKernel does not return any information in the Key Buffer parameter.

If the Get By Percentage operation is unsuccessful, the MicroKernel returns one of the following status codes:

3	The file is not open.
6	The key number parameter is invalid.
7	The key number has changed.
8	The current positioning is invalid.
9	The operation encountered the end-of-file.
22	The data buffer parameter is too short.
41	The MicroKernel does not allow the attempted operation.
43	The specified record address is invalid.
82	The MicroKernel lost positioning.

Positioning

If successful when seeking a record relative to a specified key path, the Get By Percentage operation establishes the new logical and physical currencies based respectively on the specified Key Number and the retrieved record.

If successful when seeking a record based on the record's physical location within the file, the Get By Percentage operation establishes the new physical currency based on the retrieved record.

If the Get By Percentage operation is unsuccessful, the MicroKernel changes no currency.

Get Direct/Chunk (23)

The Get Direct/Chunk operation can retrieve one or more portions, called *chunks*, of a record. This operation is especially useful on files containing records longer than 65,535 bytes. Such records are too long to be retrieved by the other Get and Step operations, due to restrictions on the length of the Data Buffer parameter. Your application specifies the record from which chunks are to be retrieved by supplying its physical address. The location of a chunk in a record is generally specified by its offset and length.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X		X
Returned		X	X	X		

Prerequisites

◆ The file must be open.

◆ You must provide the 4-byte physical location of the record. You can retrieve this location with a Get Position operation (22).

◆ You must provide a large enough Data Buffer to contain all values that a Get Direct/Chunk operation returns. The Data Buffer must also be able to contain the entire chunk descriptor (all the chunk definitions) when the Get Direct/Chunk operation is performing an indirect chunk operation. The maximum size of the Data Buffer is limited as shown in Table 2-9.

Table 2-9   Data Buffer Size Limitations by Environment

Environment	Maximum Data Buffer Size
Local calls to server or workstation engine	64,512 bytes
Remote calls via DOS Requester (BREQNT)	55,512 bytes
Remote calls via DOS Requester (BREQUEST)	57,000 bytes
Remote calls via Win16, Win32, or OS/2 Requester	65,153 bytes

Procedure

1. Set the Operation Code to 23. Optionally, you can include a lock bias:

   - 100—Single wait record lock.

   - 200—Single no-wait record lock.

   - 300—Multiple wait record lock.

   - 400—Multiple no-wait record lock.

   For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Specify a Data Buffer, as described in "Details".

4. Specify the Data Buffer Length as either the length of the input structure (Table 2-10 or Table 2-11) or the number of bytes you requested for the MicroKernel to retrieve, whichever is larger.

   Some options for the Get Direct/Chunk operation retrieve chunks to locations other than the Data Buffer. See the "Details" section for more information about calculating the Data Buffer Length.

5. Set the Key Number parameter to –2 (0xFE).

Details

Use one of the following chunk descriptors in the Data Buffer:

- ◆ Random Chunk Descriptor—To retrieve a single chunk per operation, or to retrieve multiple chunks in a single operation when the chunks are spaced randomly throughout the record.

- ◆ Rectangle Chunk Descriptor—To retrieve multiple chunks in an operation, when each chunk is the same length and chunks are spaced equidistantly in the record.

Random Chunks

The following example shows a record with three randomly spaced chunks (shaded areas): chunk 0 (bytes 0x12 through 0x16), chunk 1 (bytes 0x2A through 0x31), and chunk 2 (bytes 0x41 through 0x4E).

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

10	11							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

To fetch random chunks, you must create a structure in the Data Buffer, based on the following table.

Table 2-10   Data Buffer for Random Chunk Operations

Element	Length (in Bytes)	Description
Record Address	4	The 4-byte physical location of the record. You can retrieve this location with a Get Position operation (22).
Random Chunk Descriptor		
Subfunction	4	Type of chunk descriptor; one of the following: • 0x80000000 (Direct random chunk descriptor)—Retrieves chunks directly into the Data Buffer. The first chunk is retrieved and stored at offset 0 in the Data Buffer, the second chunk immediately follows the first, and so on. • 0x80000001 (Indirect random chunk descriptor)—Retrieves chunks into addresses specified by the Chunk Definitions.
NumChunks	4	Number of chunks to retrieve. The value must be at least 1. Although no explicit maximum value exists, the chunk descriptor must fit in the Data Buffer, which is limited in size as described in Table 2-9.

Table 2-10   Data Buffer for Random Chunk Operations *continued*

Element	Length (in Bytes)	Description
Chunk Definition (Repeat for each chunk)	12	Each Chunk Definition is a 4-byte Chunk Offset, followed by a 4-byte Chunk Length, followed by a 4-byte User Data, described as follows: ◆ Chunk Offset—Indicates where the chunk begins as an offset in bytes from the beginning of the record. The minimum value is 0, and the maximum value is the offset of the last byte in the record. ◆ Chunk Length—Indicates how many bytes are in the chunk. The minimum value is 0, and the maximum value 65,535; however, the chunk descriptor must fit in the Data Buffer, which is limited in size as described in Table 2-9. ◆ User Data—(Used only for indirect descriptors.) A 32-bit pointer to the actual chunk data. The format you should use depends on your operating system.[1] The MicroKernel ignores this element for direct chunk descriptor subfunctions.

1 For 16-bit applications, initialize User Data as a 16-bit offset and a 16-bit segment. User Data cannot address memory beyond the end of its segment. When Chunk Length is added to the offset portion of User Data, the result must be within the segment that User Data defines. By default, the MicroKernel does not check for violations of this rule and does not properly handle such violations.

The following table shows a sample Data Buffer for a fetching direct random chunks.

Element	Sample Value	Length (in Bytes)
Record Address	0x00000628	4
Subfunction	0x8000000	4
NumChunks	3	4
Chunk 0		

Element	Sample Value	Length (in Bytes)
Chunk Offset	18	4
Chunk Length	5	4
User Data	N/A	4
Chunk 1		
Chunk Offset	42	4
Chunk Length	8	4
User Data	N/A	4
Chunk 2		
Chunk Offset	65	4
Chunk Length	14	4
User Data	N/A	4

Rectangle Chunk Descriptor Structure

When chunks of the same length are spaced equidistantly throughout a record, you can describe all the chunks to retrieve with a rectangle chunk descriptor. For example, consider the following diagram, which represents offset 0x00 through 0x4F in a record:

| 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 |

The record contains three chunks (shaded areas): chunk 0 (bytes 0x19 through 0x1C), chunk 1 (bytes 0x29 through 0x2C), and chunk 2 (bytes 0x39 through 0x3C). Each chunk is four bytes long, and a total of 16 (0x10) bytes, calculated from the beginning of each chunk, separates the chunks from one another.

You can retrieve all three chunks using a single rectangle descriptor. To fetch rectangle chunks, you must create a structure in the Data Buffer based on the following table.

Table 2-11   Data Buffer for Rectangle Chunks

Element	Length (in Bytes)	Description
Record Address	4	The 4-byte physical location of the record. You can retrieve this location with a Get Position operation (22).
Rectangle Chunk Descriptor		
Subfunction	4	Type of chunk descriptor; one of the following: ◆ 0x80000002 (Direct rectangle chunk descriptor)—Retrieves chunks directly into the Data Buffer. The first chunk is retrieved and stored at offset 0 in the Data Buffer, the second chunk immediately follows the first, and so on. ◆ 0x80000003 (Indirect rectangle chunk descriptor)—Retrieves chunks into addresses specified by the User Data and Application Distance Between Rows elements.
Number of Rows	4	Number of chunks on which the rectangle chunk descriptor must operate. The minimum value is 1. No explicit maximum value exists.
Offset	4	Offset from the beginning of the record of the first byte to retrieve. The minimum value is 0, and the maximum value is the offset of the last byte in the record. If the record is viewed as a rectangle, this element refers to the offset of the first byte in the first row to be retrieved.
Bytes Per Row	4	Number of bytes to retrieve in each chunk. The minimum value is 0, and the maximum value is 65,535; however, the chunk descriptor must fit in the Data Buffer, which is limited in size as described in Table 2-9.

Table 2-11   Data Buffer for Rectangle Chunks *continued*

Element	Length (in Bytes)	Description
Distance Between Rows	4	Number of bytes between the beginning of each chunk.
User Data	4	(Used only with indirect descriptors.) A 32-bit pointer to the location into which the MicroKernel stores bytes after retrieving them from each row. The format you should use depends on your operating system.[1] The MicroKernel ignores this element for direct rectangle descriptors; however, you must still allocate the element and initialize it to 0.
Application Distance Between Rows	4	(Used only with indirect rectangle descriptors.) Number of bytes between the beginning of each chunk in the rectangle, as the rectangle is stored in your application's memory, at the address specified by User Data. The MicroKernel ignores this element for direct rectangle descriptors; however, you must still allocate the element and initialize it to 0.

1 For 16-bit applications, express User Data as a 16-bit offset followed by a 16-bit segment.

When you use an indirect descriptor, be sure that the User Data pointer is initialized so that the chunks retrieved do *not* overwrite your chunk descriptor. The MicroKernel uses the descriptor when copying the returned chunks to the locations that the User Data elements specify. In the event that you overwrite your chunk descriptor, the MicroKernel returns Status Code 62.

If the rectangle has the same number of bytes between rows when it is in memory as when it is stored as a record, set Application Distance Between Rows with the same value as Distance Between Rows. However, if the rectangle is arranged in your application's memory with either more or fewer bytes between rows, Application Distance Between Rows allows you to pass that information to the MicroKernel.

When you use an indirect rectangle descriptor, the MicroKernel uses both the User Data and the Application Distance Between Rows elements to determine the locations in which to store the data after retrieving it. The MicroKernel stores data from the first row in offset 0 of User Data. The MicroKernel stores the second row's data to an address specified by User Data plus Application Distance Between Rows. The MicroKernel stores the third row's data in the address specified by User Data plus (Application Distance Between Rows * 2), and so on.

The following table shows a sample Data Buffer for fetching a direct rectangle chunk.

Element Name	Sample Value	Length (in Bytes)
Record Address	0x00000628	4
Subfunction	0x80000002	4
Number of Rows	3	4
Offset	25	4
Bytes Per Row	4	4
Distance Between Rows	16	4
User Data	0	4
Application Distance Between Rows	0	4

Next-in-Record Subfunction Bias

If you add a bias of 0x40000000 to any of the subfunctions previously listed, the MicroKernel calculates the subfunction's Offset element values based on your physical intrarecord currency (that is, your current physical location within the record). When you use the Next-in-Record subfunction, the MicroKernel ignores the Offset element in the chunk descriptor.

Result

If the Get Direct/Chunk operation is successful and a direct chunk descriptor is used, the MicroKernel returns the chunks one after another in the Data Buffer. If you used an indirect random chunk descriptor, the MicroKernel returns the data to the locations that each chunk's User Data element specifies. If you used an indirect rectangle descriptor, the MicroKernel returns the data to locations it derives from the User Data and Application Distance Between Rows elements.

The MicroKernel also stores the total length of the chunks retrieved in the Data Buffer Length parameter. (The returned value reflects all bytes retrieved, whether they were retrieved and stored directly into the Data Buffer, or the indirect descriptor was used to retrieve and store the bytes elsewhere.) If the operation was partially successful, your application can use the value returned in the Data Buffer Length parameter to determine which chunks could not be retrieved and how many bytes of the final chunk were retrieved.

The Get Direct/Chunk operation is only partially successful if any chunk begins beyond the end of the record (resulting in the MicroKernel returning Status Code 103), or if any chunk's offset and length combine to exceed the length of the record. In the latter case, the MicroKernel returns Status Code 0 but ceases processing subsequent chunks, if any, in the operation.

> **Note**
>
> Only the Data Buffer Length parameter shows that not all of the chunks were properly retrieved. For this reason, be sure that you always check the value returned in the Data Buffer Length parameter after a Get Direct/Chunk operation.

The following status codes indicate a partially successful Get Direct/Chunk operation. When the MicroKernel returns one of these status codes, your application should check

the Data Buffer Length parameter's return value to see how much data the MicroKernel actually returned.

| 54 | The variable-length portion of the record is corrupt. |
| 103 | The chunk offset is too big. |

If the MicroKernel returns any of the following status codes, it has returned no data.

43	The specified record address is invalid.
58	The compression buffer length is too short.
62	The descriptor is incorrect.
97	The data buffer is too small.
106	The MicroKernel cannot perform a Get Next Chunk operation.

Positioning

The Get Direct/Chunk operation has no effect on logical currency. In terms of physical currency, Get Direct/Chunk makes the record from which chunks are retrieved the physical current record.

Get Direct/Record (23)

The Get Direct/Record operation retrieves a record using its physical location in the file instead of using one of the defined key paths. Use Get Direct/Record to accomplish the following:

◆ Retrieve a record faster using its physical location instead of its key value.

◆ Use the Get Position operation (22) to retrieve the 4-byte location of a record, save the location, and use Get Direct/Record to return directly to that location after performing other operations that affect currency.

◆ Use the 4-byte location to retrieve a record in a chain of duplicates without rereading all the records from the beginning of the chain.

◆ Change the current key path. A Get Position operation, followed by a Get Direct/Record operation with a different key number, establishes positioning for the current record in a different index path. A subsequent Get Next returns the next record in the file based on the new key path.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X		X
Returned		X	X	X	X	

Note

The Key Number parameter is not needed when performing a Get Direct/Record operation on a data-only file.

Prerequisites

◆ The file must be open.

◆ You must provide the 4-byte physical location of the record. You can retrieve this location with a Get Position operation (22), which returns the physical address of the current record.

Procedure

1. Set the Operation Code to 23. Optionally, you can include a lock bias:

   ◆ 100—Single wait record lock.

   ◆ 200—Single no-wait record lock.

   ◆ 300—Multiple wait record lock.

   ◆ 400—Multiple no-wait record lock.

   For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Store the 4-byte position of the requested record in the first 4 bytes of the Data Buffer.

4. Set the Data Buffer Length to a value greater than or equal to the length of the record to retrieve.

5. Set the Key Number to the key number of the path for which you want the MicroKernel to establish logical currency. Specify –1 if you do not want the MicroKernel to establish logical currency. To use the system-defined log key (also called system data), specify 125.

Result

If the Get Direct/Record operation is successful, the MicroKernel returns the requested record in the Data Buffer, the length of the record in the Data Buffer Length parameter, and the key value for the specified key path in the Key Buffer.

If the Get Direct/Record operation is unsuccessful and the MicroKernel cannot return the requested record, the MicroKernel returns one of the following status codes:

    22   The data buffer parameter is too short. (Logical currency is still established.)

    43   The specified record address is invalid. (Logical currency is not established.)

    44   The specified key path is invalid. (Logical currency is not established.)

    82   The MicroKernel lost positioning. (Logical currency is not established.)

Positioning

The Get Direct/Record operation erases any existing logical currency information and establishes the new logical currency according to the Key Number specified. It has no effect on the physical currency information.

Get Directory (18)

The Get Directory operation returns the current directory for a specified logical disk drive.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X					X
Returned					X	

Prerequisites

Your application can issue a Get Directory operation at any time. The Key Buffer should be at least 65 characters long.

Procedure

1. Set the Operation Code to 18.

2. Store the logical disk drive number in the Key Number parameter. Specify the drive as 1 for A, 2 for B, and so on. To use the default drive, specify 0.

Result

The MicroKernel returns the current directory, terminated by a binary 0, in the Key Buffer.

Positioning

The Get Directory operation has no effect on any file currency information.

Get Equal (5)

The Get Equal operation retrieves a record that has a key value equal to that specified in the Key Buffer. If the key allows duplicates, this operation retrieves the first record (chronologically) of a group with the same key values. You can use the Get Key (+50) bias to detect the presence of a value in a file. A Get Key operation is generally faster.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X		X	X	X
Returned		X	X	X		

Prerequisites

◆ The file must be open.

◆ The file cannot be a data-only file.

Procedure

1. Set the Operation Code to 5. Optionally, you can include a lock bias:

* 100—Single wait record lock.

* 200—Single no-wait record lock.

* 300—Multiple wait record lock.

* 400—Multiple no-wait record lock.

For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Set the Data Buffer Length to a value greater than or equal to the length of the record to retrieve.

4. Specify the desired key value in the Key Buffer.

5. Set the Key Number to the correct key path. To use the system-defined log key (also called system data), specify 125.

Result

If the Get Equal operation is successful, the MicroKernel returns the requested record in the Data Buffer and the length of the record in the Data Buffer Length parameter.

If the Get Equal operation is unsuccessful, the MicroKernel returns one of the following status codes:

3	The file is not open.
4	The application cannot find the key value.
6	The key number parameter is invalid.
22	The data buffer parameter is too short.

Positioning

The Get Equal operation establishes the complete logical and physical currencies and makes the retrieved record the current one.

Get First (12)

The Get First operation retrieves the logical first record based on the specified key. You can use the Get Key (+50) bias to detect the presence of a value in a file. A Get Key operation is generally faster.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X		X		X
Returned		X	X	X	X	

Prerequisites

◆ The file must be open.

◆ The file cannot be a data-only file.

Procedure

1. Set the Operation Code to 12. Optionally, you can include a lock bias:

   • 100—Single wait record lock.

   • 200—Single no-wait record lock.

   • 300—Multiple wait record lock.

   • 400—Multiple no-wait record lock.

   For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Set the Data Buffer Length to a value greater than or equal to the length of the record to retrieve.

   4. Indicate the Key Number for the key path. To use the system-defined log key (also called system data), specify 125.

Result

If the Get First operation is successful, the MicroKernel returns the requested record in the Data Buffer, stores the corresponding key value in the Key Buffer, and returns the length of the record in the Data Buffer Length parameter.

If the Get First operation is unsuccessful, the MicroKernel returns one of the following status codes:

   3   The file is not open.

   6   The key number parameter is invalid.

   9   The operation encountered the end-of-file.

   22   The data buffer parameter is too short.

Positioning

The Get First operation establishes the complete logical and physical currencies and makes the retrieved record the current one. The logical previous position points beyond the beginning of the file.

Get Greater (8)

The Get Greater operation retrieves a record in which the field specified by the Key Number has the next greater value than the one in the Key Buffer. If the key allows duplicates, this operation retrieves the first record (chronologically) of the group with the same key values. You can use the Get Key (+50) bias to detect the presence of a value in a file. A Get Key operation is generally faster.

> **Note**
>
> If you are using the Get Greater operation on descending keys, the next greater value refers to a value lower than the one specified in the Key Buffer.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X		X	X	X
Returned		X	X	X	X	

Prerequisites

◆ The file must be open.

◆ The file cannot be a data-only file.

Procedure

1. Set the Operation Code to 8. Optionally, you can include a lock bias:

   • 100—Single wait record lock.

   • 200—Single no-wait record lock.

- ◆ 300—Multiple wait record lock.

- ◆ 400—Multiple no-wait record lock.

   For more information about locking, refer to the *Btrieve Programmer's Guide*.

   2. Pass the Position Block for the file.

   3. Set the Data Buffer Length to a value greater than or equal to the length of the record you want to retrieve.

   4. Specify the desired key value in the Key Buffer parameter.

   5. Set the Key Number parameter to correspond to the correct key path. To use the system-defined log key (also called system data), specify 125.

Result

If the Get Greater operation is successful, the MicroKernel stores the record in the Data Buffer, the key value in the Key Buffer, and the length of the record in the Data Buffer Length parameter.

If the Get Greater operation is unsuccessful, the MicroKernel returns one of the following status codes:

   3   The file is not open.

   6   The key number parameter is invalid.

   22   The data buffer parameter is too short.

Positioning

The Get Greater operation establishes the complete logical and physical currencies and makes the retrieved record the current one.

Get Greater Than or Equal (9)

The Get Greater Than or Equal operation retrieves a record in which the value for the key specified by the Key Number is equal to or greater than the value you supply in the Key Buffer. The MicroKernel first tries to satisfy the equal requirement. If the key allows duplicates, this operation retrieves the first record (chronologically) of the group with the same key values. You can use the Get Key (+50) bias to detect the presence of a value in a file. A Get Key operation is generally faster.

> **Note**
>
> If you are using the Get Greater Than or Equal operation on descending keys, the next greater value refers to a value lower than the one specified in the Key Buffer.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X		X	X	X
Returned		X	X	X	X	

Prerequisites

- ◆ The file must be open.
- ◆ The file cannot be a data-only file.

Procedure

1. Set the Operation Code to 9. Optionally, you can include a lock bias:
   - • 100—Single wait record lock.
   - • 200—Single no-wait record lock.

- ◆ 300—Multiple wait record lock.
- ◆ 400—Multiple no-wait record lock.

  For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Set the Data Buffer Length to a value greater than or equal to the length of the record you want to retrieve.

4. Specify the key value in the Key Buffer parameter.

5. Set the Key Number parameter to correspond to the correct key path. To use the system-defined log key (also called system data), specify 125.

Result

If the Get Greater Than or Equal operation is successful, the MicroKernel stores the record in the Data Buffer, the key value in the Key Buffer, and the length of the record in the Data Buffer Length parameter.

If the Get Greater Than or Equal operation is unsuccessful, the MicroKernel returns one of the following status codes:

3	The file is not open.
6	The key number parameter is invalid.
22	The data buffer parameter is too short.

Positioning

The Get Greater Than or Equal operation establishes the complete logical and physical currencies and makes the retrieved record the current one.

Get Key (+50)

The Get Key bias allows you to perform a Get operation without actually retrieving a data record. You can use Get Key to detect the presence of a value in a file. A Get Key operation is generally faster than its corresponding Get operation. You can use the Get Key operation with any of the following Get operations:

- Get Equal (5)
- Get Next (6)
- Get Previous (7)
- Get Greater (8)
- Get Greater Than or Equal (9)
- Get Less Than (10)
- Get Less Than or Equal (11)
- Get First (12)
- Get Last (13)

Parameters

The parameters are the same as those for the corresponding Get operation, except that the MicroKernel ignores the Data Buffer Length and does not return a record in the Data Buffer.

Prerequisites

The prerequisites for a Get Key operation are the same as those for the corresponding Get operation.

Procedure

1. Set the parameters as you would for the corresponding Get operation. You do not need to initialize the Data Buffer Length.

2. Set the Operation Code to the Get operation you want to perform, plus 50. For example, to perform a Get Key (+50) with the Get Equal operation (5), set the Operation code to 55.

The MicroKernel does not allow Delete or Update operations after a Get Key operation (+50). Before the MicroKernel performs Update or Delete operations, it compares the current usage count of the data page it intends to modify with the usage count of the data page when the record was read. To obtain the usage count, the MicroKernel must read the data page.

Because the Get Key operation does not read the data page, no usage count is available for comparison on the Update or Delete. The Update or Delete fails because the MicroKernel cannot perform its passive concurrency conflict checking without the compare. When the Update or Delete fails, the MicroKernel returns Status Code 8.

Result

If the MicroKernel finds the requested key, it returns the key value in the Key Buffer and Status Code 0. Otherwise, the MicroKernel returns a status code indicating why it cannot find the key value.

Positioning

The Get Key operation establishes the current positioning in a similar manner to the corresponding Get operation. However, when a Get Key operation involves a key that allows duplicates, the MicroKernel ignores the duplicate instances of the current retrieved key value. After a Get Key operation, the logical previous position points to the record

containing the previous lesser key value. The logical next position points to the record with the next greater key value.

For example, assume you perform a Get Key/Get Equal operation (55) on a last name key that contains eight occurrences of Smith and a single Smythe. The logical next position does not point to the next Smith, but to Smythe.

Because a Get Key operation does not positively identify any one record, the MicroKernel does not allow an Update or Delete operation to follow a Get Key operation.

Get Last (13)

The Get Last operation retrieves the logical last record based on the specified key. If duplicates exist for the last key value, the record returned is the last duplicate. You can use the Get Key (+50) bias to detect the presence of a value in a file. A Get Key operation is generally faster.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X		X		X
Returned		X	X	X	X	

Prerequisites

◆ The file must be open.

◆ The file cannot be a data-only file.

Procedure

1. Set the Operation Code to 13. Optionally, you can include a lock bias:

   ◆ 100—Single wait record lock.

   ◆ 200—Single no-wait record lock.

   ◆ 300—Multiple wait record lock.

   ◆ 400—Multiple no-wait record lock.

   For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Set the Data Buffer Length to a value greater than or equal to the length of the record you want to retrieve.

4. Specify the Key Number for the key path. To use the system-defined log key (also called system data), specify 125.

Result

If the Get Last operation is successful, the MicroKernel returns the requested record in the Data Buffer, stores the corresponding key value in the Key Buffer, and returns the length of the record in the Data Buffer Length parameter.

If the Get Last operation is unsuccessful, the MicroKernel returns one of the following status codes:

3	The file is not open.
6	The key number parameter is invalid.
9	The operation encountered the end-of-file.
22	The data buffer parameter is too short.

Positioning

The Get Last operation establishes the complete logical and physical currencies and makes the retrieved record the current one. The logical next position points beyond the end of the file.

Get Less Than (10)

The Get Less Than operation retrieves a record in which the value for the key specified by the Key Number has the previous lesser value than the value you supply in the Key Buffer. If the key allows duplicate values, this operation retrieves the last record (chronologically) of the group with the same key values. You can use the Get Key (+50) bias to detect the presence of a value in a file. A Get Key operation is generally faster.

> **Note**
>
> If you are using the Get Less Than operation on descending keys, the next lesser value refers to a value higher than the one specified in the Key Buffer.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X		X	X	X
Returned		X	X	X	X	

Prerequisites

◆ The file must be open.

◆ The file cannot be a data-only file.

Procedure

1. Set the Operation Code to 10. Optionally, you can include a lock bias:

   • 100—Single wait record lock.

   • 200—Single no-wait record lock.

- 300—Multiple wait record lock.

- 400—Multiple no-wait record lock.

For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Set the Data Buffer Length to a value greater than or equal to the length of the record you want to retrieve.

4. Specify the desired key value in the Key Buffer parameter.

5. Set the Key Number parameter to the key path. To use the system-defined log key (also called system data), specify 125.

Result

If the Get Less Than operation is successful, the MicroKernel returns the record in the Data Buffer, the key value for the record in the Key Buffer, and the length of the record in the Data Buffer Length parameter.

If the Get Less Than operation is unsuccessful, the MicroKernel returns one of the following status codes:

3	The file is not open.
6	The key number parameter is invalid.
22	The data buffer parameter is too short.

Positioning

The Get Less Than operation establishes the complete logical and physical currencies and makes the retrieved record the current one.

Get Less Than or Equal (11)

The Get Less Than or Equal operation retrieves a record in which the value for the key specified by the Key Number has an equal or a previous lesser value than the value you supply in the Key Buffer. The MicroKernel first tries to satisfy the equal requirement. If the key allows duplicate values, this operation retrieves the last record (chronologically) of the group with the same key values. You can use the Get Key (+50) bias to detect the presence of a value in a file. A Get Key operation is generally faster.

> **Note**
>
> If you are using the Get Less Than or Equal operation on descending keys, the next lesser value refers to a value higher than the one specified in the Key Buffer.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X		X	X	X
Returned		X	X	X	X	

Prerequisites

- ◆ The file must be open.
- ◆ The file cannot be a data-only file.

Procedure

1. Set the Operation Code to 11. Optionally, you can include a lock bias:
   - ◆ 100—Single wait record lock.
   - ◆ 200—Single no-wait record lock.

- 300—Multiple wait record lock.
- 400—Multiple no-wait record lock.

For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Set the Data Buffer Length to a value greater than or equal to the length of the record you want to retrieve.

4. Specify the key value in the Key Buffer parameter.

5. Set the Key Number parameter to the key path. To use the system-defined log key (also called system data), specify 125.

Result

If the Get Less Than or Equal operation is successful, the MicroKernel returns the record in the Data Buffer, the key value for the record in the Key Buffer, and the length of the record in the Data Buffer Length parameter.

If the Get Less Than or Equal operation is unsuccessful, the MicroKernel returns one of the following status codes:

3	The file is not open.
6	The key number parameter is invalid.
22	The data buffer parameter is too short.

Positioning

The Get Less Than or Equal operation establishes the complete logical and physical currencies and makes the retrieved record the current one.

Get Next (6)

The Get Next operation retrieves the record in the logical next position (based on the specified key). You can use the Get Next operation to retrieve records within a group of records that have duplicate key values. You can use the Get Key (+50) bias to detect the presence of a value in a file. A Get Key operation is generally faster.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X		X	X	X
Returned		X	X	X	X	

Prerequisites

◆ The file must be open.

◆ The file cannot be a data-only file.

◆ Your application must have an established logical next position based on the specified key.

Procedure

1. Set the Operation Code to 6. Optionally, you can include a lock bias:

◆ 100—Single wait record lock.

◆ 200—Single no-wait record lock.

◆ 300—Multiple wait record lock.

◆ 400—Multiple no-wait record lock.

For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Set the Data Buffer Length to a value greater than or equal to the length of the record you want to retrieve.

4. Specify the key value from the previous operation in the Key Buffer.

Pass the Key Buffer exactly as the MicroKernel returned it on the previous call, because the MicroKernel may need the information previously stored there to determine its current position in the file.

5. Set the Key Number parameter to the key path used on the previous call. You cannot change key paths using a Get Next operation.

Result

If the Get Next operation is successful, the MicroKernel returns the record in the Data Buffer, the key value for the record in the Key Buffer, and the length of the record in the Data Buffer Length parameter.

If the Get Next operation is unsuccessful, the MicroKernel returns one of the following status codes:

3	The file is not open.
6	The key number parameter is invalid.
7	The key number has changed.
8	The current positioning is invalid.
9	The operation encountered the end-of-file.
22	The data buffer parameter is too short.
82	The MicroKernel lost positioning.

The operation returns Status Code 9 if the logical next position points beyond the end of the file.

Positioning

The Get Next operation establishes the complete logical and physical currencies and makes the retrieved record the current one.

Get Next Extended (36)

The Get Next Extended operation examines one or more records, starting at the logical next position and proceeding toward the end of the file, based on the specified key. It checks to see if the examined record or records satisfy a filtering condition, and it retrieves the ones that do. The filtering condition is a logic expression and is not limited to key fields only.

Get Next Extended can also extract specified portions from records and return only those portions to an application.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X	X	X
Returned		X	X	X	X	

Prerequisites

◆ The file must be open.

◆ The file cannot be a data-only file.

◆ You must have an established logical next position based on the specified key.

Procedure

1. Set the Operation Code to 36. Optionally, you can include a lock bias:

  ◆ 100—Single wait record lock.

  ◆ 200—Single no-wait record lock.

- ◆ 300—Multiple wait record lock.

- ◆ 400—Multiple no-wait record lock.

  For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Specify a large enough Data Buffer to accommodate either the input Data Buffer or the returned Data Buffer, whichever is larger. Initialize the Data Buffer according to the structure shown in Table 2-12.

4. Specify the Data Buffer Length as either the length of the input structure (Table 2-12) or the length of the returned structure (Table 2-13), whichever is larger.

  The MicroKernel sets up a buffer as a workspace for extended operations. You configure the size of this buffer using the Extended Operation Buffer Size option. The sum of the Data Buffer structure, plus the longest record to be retrieved, plus 355 bytes of requester overhead, cannot exceed the configured buffer size. (Requester overhead is not applicable for DOS workstation engines.)

5. Specify the key value from the previous operation in the Key Buffer. Pass the Key Buffer exactly as the MicroKernel returned it on the previous call, because the MicroKernel may need the information previously stored there to determine its current position in the file.

6. Set the Key Number parameter to the key path used on the previous call. You cannot change key paths using a Get Next Extended operation.

Details

The following table shows the structure of the input data buffer.

Table 2-12   Input Data Buffer Structure for Extended Get and Step Operations

Element	Length (in Bytes)	Description
Header	2	Total length of the Data Buffer.
	2	One of two non null terminated string constant values: "EG"—Begin with the record after the one at which you are positioned. "UC"—Begin with the record at which you are positioned. For Step Next Extended operations, always set this value to "EG".
Filter (Fixed Portion)	2	Maximum Reject Count, which is the number of records that the MicroKernel can skip while searching for records that satisfy the filter condition. You can set the value from 0 to 65,535. (0 means the MicroKernel uses the system-defined maximum reject count, which is 4,095.)
	2	Number of Terms in the logic expression of the filter condition. (0 means the MicroKernel performs no filtering.)

Table 2-12   Input Data Buffer Structure for Extended Get and Step Operations *continued*

Element	Length (in Bytes)	Description
Filter (Repeating Portion—one for each term of logic expression)	1	Data Type of the field. Use one of the codes shown in <u>Table 2-4</u>.
	2	Field Length.
	2	Field Offset (zero relative).
	1	Specifies a Comparison Code: 1–Equal 2–Greater than 3–Less than 4–Not equal 5–Greater than or equal 6–Less than or equal Add a +8 bias to compare strings using one of the file's existing ACSs.[1] Add a +32 bias to compare strings using the file's default ACS, which is the first ACS defined in the file.
		Add a +64 bias if the second operand is another field of the record, rather than a constant. Add a +128 bias to compare strings without case sensitivity.

Element	Length (in Bytes)	Description
Filter (Repeating Portion *continued*)	1	Indicates an AND/OR logic expression: 0—Identifies the last term 1—Next term is connected with AND 2—Next term is connected with OR
	2 or n	When comparing two fields: a 2-byte, zero-relative offset to the second field. (The second field must be the same type and length.) *or* When comparing a field to a constant: the actual value of the constant. The length of the constant (n) must equal the length of the field.
	0, 5, 9, or 17	When specifying an ACS by name (bias +8), the ACS identifier using one of the name formats shown in Table 2-8.
Descriptor (Fixed Portion)	2	Number of Records to retrieve. To retrieve only one record instead of a set of records, specify 1.
	2	Number of Fields to extract from each record.
Descriptor (Repeating Portion—one for each extracted field)	2	Field Length to extract.
	2	Field Offset (zero relative).

1 If you use both a +8 bias and a +32 bias, the +32 bias is ignored.

The MicroKernel interprets the AND and OR operators used in a filter with the extended Get and Step operations in strict left-to-right order. The MicroKernel evaluates an expression in the filter and proceeds as follows:

- ◆ If the expression is true when applied to the current record and the next operator is OR, the MicroKernel accepts this record as meeting the filter condition.

- ◆ If the expression is true and the next operator is AND, the MicroKernel continues to evaluate each expression until one of the following situations occurs:

  - • The MicroKernel reaches an OR expression.

  - • One of the expressions evaluates to false.

  - • The MicroKernel reaches the end of the filter.

- ◆ If the expression is false and the next operator is OR, the MicroKernel continues and evaluates the next expression in the filter.

- ◆ If the expression is false and the next operator is AND, the MicroKernel rejects the record.

The search for records stops if any one of the following conditions is met:

- ◆ The MicroKernel finds the requested number of records that satisfy the filter.

- ◆ While the MicroKernel searches for records to satisfy the filter condition, the number of records it examines exceeds the Maximum Reject Count you specify.

- ◆ The current key path is used as a filtering field and the MicroKernel reaches a rejected record after which no records can satisfy the filtering condition in the rest of the file.

- ◆ The MicroKernel reaches the end of the file.

Examples

To get the next entire record that satisfies the filtering condition, set the filter portion as desired and set the descriptor fields as follows:

1. Set the Number of Records to 1.

2. Set the Number of Fields to 1.

3. Set the Field Length to the length of the entire record to retrieve.

4. Set the Field Offset to 0.

To retrieve the next 12 records without using a filtering condition and extract 4 fields from each record, set the filter Number of Terms to 0 and set the descriptor fields as follows:

1. Set the Number of Records to 12.

2. Set the Number of Fields to 4.

3. Set the Field Length and Field Offset parameters for each of the 4 fields extracted.

Retrieving Fields from Records

When retrieving one or more fields (portions) of records with an extended operation, you must ensure that the Data Buffer can accommodate the information that the operation returns.

The following table illustrates the structure of the Data Buffer that the MicroKernel returns.

Table 2-13   Returned Data Buffer Structure for Extended Get and Step Operations

Element	Length (in Bytes)	Description
Number of Records	2	Number of records returned.
Repeating portion (one for each record retrieved)		
Length 0	2	Length of the first record image (all fields combined).
Position 0	4	Physical currency (address) of the first record.
Record 0	n	Image of the first record (all fields combined).
. . .		
Length x	2	Length in bytes of the last record image (all fields combined).
Position x	4	Physical currency (address) of the last record.
Record x	n	Image of the last record (all fields combined).

If all returned records (or fields of records) are fixed length, your application can easily calculate the location of data within the returned Data Buffer. However, your application may need to perform extra steps to extract the variable-length portion of records from the Data Buffer that an extended operation returns.

The MicroKernel does not pad any record image in the returned Data Buffer when returning the variable-length portion of a record. Consequently, if you allow room in the returned Data Buffer for the maximum number of bytes that the variable-length portion of a record could occupy, but the actual data returned is less than that maximum, the MicroKernel starts the field description for the next returned field immediately following the data for the current field.

For example, suppose your fixed-record length is 100 bytes, your variable-length portion is up to 300 bytes, and you want to return just the variable-length portion of 5 records.

You would use the descriptor element of the input buffer to set a Field Length of 300 and a Field Offset of 100. For the returned buffer, you need 2 bytes for the Number of Records plus 306 bytes for each record (that is, 2 bytes for the length, 4 bytes for the address, and 300 bytes for the data), as shown in the following calculation:

```
2 + ((2 bytes + 4 bytes + 300 bytes) * 5) = 1532 bytes
```

However, suppose that the variable-length portion of the first record returned contains only 50 bytes of data. This means the 2-byte length for the second record returned is stored at offset 58 in the Data Buffer, immediately following the image of the first record's field. In such a situation, your application must parse the length, position, and data from the Data Buffer that the MicroKernel returns.

Result

If the Get Next Extended operation is successful, the MicroKernel returns the following:

◆ In the Data Buffer, one or more fields from one or more records. (See Table 2-13.)

◆ In the Data Buffer Length, the total number of bytes received.

◆ In the Key Buffer, the key value for the last data record received.

If the Get Next Extended operation is unsuccessful, the MicroKernel returns one of the following status codes:

3	The file is not open.
6	The key number parameter is invalid.
7	The key number has changed.
8	The current positioning is invalid.
9	The operation encountered the end-of-file.

22	The data buffer parameter is too short.
60	The specified reject count has been reached.
61	The work space is too small.
62	The descriptor is incorrect.
64	The filter limit has been reached.
65	The field offset is incorrect.
82	The MicroKernel lost positioning.
134	The MicroKernel cannot read the International Sorting Rule.
135	The specified International Sort Rule table is corrupt or otherwise invalid.
136	The MicroKernel cannot find the specified Alternate Collating Sequence in the file.

It is possible for the MicroKernel to return a nonzero status code and also return valid data in the Data Buffer. In this case, the last record returned may be incomplete. If the Data Buffer Length parameter returned is greater than 0, check the Data Buffer for extracted data.

If a field can only be partially filled because the record is too short, then the MicroKernel returns what it can of the record to and including the partial field. If the partial field is the last field to be extracted, then the MicroKernel continues the operation. Otherwise, the MicroKernel aborts the operation and returns a Status Code 22.

For example, consider a Get Next Extended operation that retrieves 3 fields from 2 variable-length records. The first record is 55 bytes long and the second is 50 bytes long. The 3 fields to be retrieved are defined as follows:

- ◆ Field 1 begins at offset 2 and is 2 bytes long.
- ◆ Field 2 begins at offset 45 and is 10 bytes long.
- ◆ Field 3 begins at offset 6 and is 2 bytes long.

When the MicroKernel performs the Get Next Extended operation, it returns the first record without any problem. However, when attempting to extract field 2's 10 bytes from the second record, the MicroKernel finds that only 5 bytes are available (between offset 45 and the end of the record, at offset 49). At this point, the MicroKernel does not pad the missing 5 bytes of field 2, and thus cannot extract field 3. Instead, the MicroKernel returns Status Code 22 and places all of field 1 and the first 5 bytes of field 2 in the return Data Buffer.

Depending on the fields and the operators used in the filtering condition, the MicroKernel may be able to optimize your request. After reaching a certain rejected record, it returns Status Code 64, indicating that no records can satisfy the filtering conditions in the rest of the file.

Positioning

The Get Next Extended operation establishes the complete logical and physical currencies. The last record examined becomes the current record. This record can be either a record that satisfies the filtering condition and is retrieved, or a record that does not satisfy the filtering condition and is rejected.

> **Note**
>
> The MicroKernel does not allow Delete or Update operations after a Get Next Extended operation. Because the current record is the last record examined, there is no way to ensure that your application would delete or update the intended record.

Get Position (22)

The Get Position operation returns the physical 4-byte position of the current record. Get Position fails if there is no established physical currency when you issue the operation. Once you determine a record's position (address), you can use the Get Direct/Record operation (23) to retrieve that record directly by its physical location in the file.

The MicroKernel does not perform any disk I/O to process a Get Position request.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X		X		X
Returned		X	X	X		

Prerequisites

- ◆ The file must be open.
- ◆ Your application must have established physical currency.

Procedure

1. Set the Operation Code to 22.
2. Pass the Position Block for the file.
3. Set the Data Buffer Length to at least 4 bytes.
4. Set the Key Number to 0.

Result

If the Get Position operation is successful, the MicroKernel returns the position of the record in the Data Buffer. The position is a 4-byte binary value (most significant word first) that indicates the record's offset (in bytes) into the file. The MicroKernel also sets the Data Buffer Length to 4 bytes.

If the Get Position operation is unsuccessful, the MicroKernel returns one of the following status codes:

3	The file is not open.
8	The current positioning is invalid.

Positioning

The Get Position operation has no effect on positioning.

Get Previous (7)

The Get Previous operation retrieves the record in the logical previous position based on a specified key. You can use the Get Previous operation to retrieve a record within a group of records that have duplicate key values. You can use the Get Key (+50) bias to detect the presence of a value in a file. A Get Key operation is generally faster.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X		X	X	X
Returned		X	X	X	X	

Prerequisites

- ◆ The file must be open.

- ◆ The file cannot be a data-only file.

- ◆ Your application must have established a logical previous position based on the specified key.

Procedure

1. Set the Operation Code to 7. Optionally, you can include a lock bias:

   - 100—Single wait record lock.

   - 200—Single no-wait record lock.

   - 300—Multiple wait record lock.

   - 400—Multiple no-wait record lock.

For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Set the Data Buffer Length to a value greater than or equal to the length of the record you want to retrieve.

4. Specify the key value from the previous operation in the Key Buffer. Pass the Key Buffer exactly as the MicroKernel returned it on the previous call. The MicroKernel may need the information previously stored in the Key Buffer to determine its current position in the file.

5. Set the Key Number parameter to the key path used on the previous call. You cannot change key paths using a Get Previous operation.

Result

If the Get Previous operation is successful, the MicroKernel updates the Key Buffer with the key value for the previous record, returns the previous record in the Data Buffer, and returns the length of the record in the Data Buffer Length parameter.

If the Get Previous operation is unsuccessful, the MicroKernel returns one of the following status codes:

3	The file is not open.
6	The key number parameter is invalid.
7	The key number has changed.
8	The current positioning is invalid.
9	The operation encountered the end-of-file.
22	The data buffer parameter is too short.
82	The MicroKernel lost positioning.

This operation returns Status Code 9 if the logical previous position points beyond the beginning of the file.

Positioning

The Get Previous operation establishes the complete logical and physical currencies and makes the retrieved record the current one.

Get Previous Extended (37)

The Get Previous Extended operation examines one or more records, starting at the logical previous position and proceeding toward the beginning of the file, based on the specified key. It checks to see if the examined record or records satisfy a filtering condition, and it retrieves the ones that do. The filtering condition is a logic expression and is not limited to key fields only.

Get Previous Extended can also extract specified portions of records and return only those portions to an application.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X	X	X
Returned		X	X	X	X	

Prerequisites

- ◆ The file must be open.

- ◆ The file cannot be a data-only file.

- ◆ You must have an established logical previous position based on the specified key.

Procedure

1. Set the Operation Code to 37. Optionally, you can include a lock bias:

   - 100—Single wait record lock.

   - 200—Single no-wait record lock.

   - 300—Multiple wait record lock.

   - 400—Multiple no-wait record lock.

   For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Specify a large enough Data Buffer to accommodate either the input Data Buffer or the returned Data Buffer, whichever is larger. Initialize the Data Buffer according to the structure shown in Table 2-12.

4. Specify the Data Buffer Length as either the length of the input structure (Table 2-12) or the length of the returned structure (Table 2-13), whichever is larger.

   The MicroKernel sets up a buffer as a workspace for extended operations. You configure the size of this buffer using the Extended Operation Buffer Size option. The sum of the Data Buffer structure, plus the longest record to be retrieved, plus 355 bytes of requester overhead, cannot exceed the configured buffer size. (Requester overhead is not applicable in DOS workstation engines.)

5. Specify the key value from the previous operation in the Key Buffer. Pass the Key Buffer exactly as the MicroKernel returned it on the previous call, because the MicroKernel may need the information previously stored there to determine its current position in the file.

6. Set the Key Number parameter to the key path used on the previous call. You cannot change key paths using a Get Previous Extended operation.

Details

This operation uses the same input and returned Data Buffers as the Get Next Extended operation. Refer to "Details" for more information.

Result

This operation returns the same results as the Get Next Extended operation. Refer to "Result" for more information.

Positioning

The Get Previous Extended operation establishes the complete logical and physical currencies. The last record examined becomes the current record. This record can be either a record that satisfies the filtering condition and is retrieved, or a record that does not satisfy the filtering condition and is rejected.

> **Note**
>
> The MicroKernel does not allow Delete or Update operations after a Get Previous Extended operation. Because the current record is the last record examined, there is no way to ensure that your application would delete or update the intended record.

Insert (2)

The Insert operation inserts a record into a file. The MicroKernel adjusts the B-trees for the keys to reflect the key values for the new record.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X		X
Returned		X			X	

Prerequisites

◆ The file must be open.

◆ The record to be inserted must be the proper length, and the key values must conform to the keys defined for the file.

Procedure

1. Set the Operation Code to 2.

2. Pass the Position Block for the file.

3. In the Data Buffer, store the record to be inserted.

4. Specify the Data Buffer Length. This value must be at least as long as the fixed-length portion of the record.

5. Specify the Key Number that the MicroKernel uses to establish positioning information (currency). To use the NCC option, specify –1 (0xFF) for the Key Number. To use the system-defined log key (also called system data), specify 125.

Result

If the Insert operation is successful, the MicroKernel places the new record in the file, updates the B-trees for the keys to reflect the new record, and returns the value of the specified key in the Key Buffer. If you insert a record that contains an AUTOINCREMENT key value initialized to binary 0, the MicroKernel also returns the inserted record in the Data Buffer, including the AUTOINCREMENT value assigned by the MicroKernel. An NCC Insert operation does not change the value of the Key Buffer parameter.

If the Insert operation is unsuccessful, the MicroKernel returns one of the following status codes:

2	The application encountered an I/O error.
3	The file is not open.
5	The record has a key field containing a duplicate key value.
18	The disk is full.
21	The key buffer parameter is too short.
22	The data buffer parameter is too short.

Positioning

An Insert operation that does not specify the NCC option establishes the complete logical and physical currencies and makes the inserted record the current one. The logical currency is based on the specified key.

An NCC Insert operation establishes physical currency without affecting logical currency. This means that an application, having performed an NCC Insert operation, has the same logical position in the file as it had prior to the Insert operation. In such a situation, operations that follow an NCC Insert—such as Get Next (6), Get Next Extended (36), Get Previous (7), and Get Previous Extended (37)—return values based on the application's logical currency prior to the NCC Insert.

> **Note**
>
> The MicroKernel does not return any information in the Key Buffer parameter as the result of an NCC Insert operation. Therefore, an application that must maintain the logical currency must not change the value of the Key Buffer following the NCC Insert operation. Otherwise, the next Get operation has unpredictable results.

The MicroKernel establishes the physical currency to a newly inserted record for both the standard Insert and the NCC Insert operations. Operations following an NCC Insert operation—such as Step Next (24), Step Next Extended (38), Step Previous (35), Step Previous Extended (39), Update (3), Delete (4), and Get Position (22)—operate based on the new physical currency.

Insert Extended (40)

The Insert Extended operation inserts one or more records into a file. The MicroKernel adjusts the B-trees for the keys to reflect the key values for the new records.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X		X
Returned		X	X		X	

> **Note**
>
> When using the no-currency-change (NCC) option, the Insert Extended operation does not update the value of the Key Buffer parameter; it does not return any information in that parameter.

Prerequisites

- ◆ The file must be open.
- ◆ The records to be inserted must be the proper length, and the key values must conform to the keys defined for the file.

Procedure

1. Set the Operation Code to 40.
2. Pass the Position Block for the file.
3. Specify the Data Buffer according to the structure shown in Table 2-14.

4. Specify the Data Buffer Length. This value must be exactly the size of the Data Buffer structure.

5. Specify the Key Number that the MicroKernel uses to establish currency. To use the NCC option, specify –1 (0xFF) for the Key Number. To use the system-defined log key (also called system data), specify 125.

Details

The following table shows the data buffer structure.

Table 2-14   Data Buffer Structure for the Insert Extended Operation

Element	Length (in Bytes)	Description
Fixed portion	2	Number of records inserted.
Repeating portion (one for each record)		
	2	Length of the record image.
	n	Record image.

Result

If the Insert Extended operation is successful, the MicroKernel places the new records in the file, updates all the B-trees to reflect the new records that were inserted, and (except for NCC Insert Extended operation) returns in the Key Buffer the value of the specified key from the last record inserted. In addition, in the first word of the returned Data Buffer, the MicroKernel places the number of records that were successfully inserted into the file. Following the first word of the Data Buffer, the MicroKernel stores the addresses of the inserted records.

If the operation is only partially successful and the MicroKernel returns a nonzero status code, the first word of the Data Buffer equals the number of records that were

successfully inserted. The record that caused the error is the number of records that were successfully inserted plus one.

If the Insert Extended operation is unsuccessful, the MicroKernel returns one of the following status codes:

2	The application encountered an I/O error.
3	The file is not open.
5	The record has a key field containing a duplicate key value.
18	The disk is full.
21	The key buffer parameter is too short.
22	The data buffer parameter is too short.

Positioning

An Insert Extended operation that does not specify the NCC option establishes the complete logical and physical currencies and makes the last inserted record the current one (unless the inserted record's key value is null). The logical currency is based on the specified key.

An NCC Insert Extended operation establishes physical currency without affecting logical currency. This means that an application, having performed an NCC Insert Extended operation, has the same logical position in the file as it had prior to the operation. In such a situation, operations that follow an NCC Insert Extended operation—such as Get Next (6), Get Next Extended (36), Get Previous (7), and Get Previous Extended (37)—return values based on the application's logical currency prior to the NCC Insert Extended operation.

Note

The MicroKernel does not return any information in the Key Buffer parameter as the result of an NCC Insert Extended operation. Therefore, an application that must maintain the logical currency must not change the value of the Key Buffer following the NCC Insert Extended operation. Otherwise, the next Get operation has unpredictable results.

The MicroKernel establishes the physical currency to a newly inserted record for both the standard Insert Extended and the NCC Insert Extended operations. Therefore, operations following an NCC Insert Extended operation—such as Step Next (24), Step Next Extended (38), Step Previous (35), Step Previous Extended (39), Update (3), Delete (4), and Get Position (22)—operate based on the new physical currency.

An NCC Insert Extended operation is useful when an application must save its logical position in the file prior to executing the Insert Extended operation in order to perform another operation based on the original logical currency, such as a Get Next operation (6).

To achieve this effect without an NCC Insert Extended operation, your application would have to execute the following steps:

1. Get Position (22)—Obtains the 4-byte physical address for the logical current record. The application saves this value and passes it back in Step 3.

2. Insert Extended (40)—Inserts the new records. This operation establishes new logical and physical currencies.

3. Get Direct/Record (23)—Re-establishes logical and physical currencies as they were in Step 1.

The NCC Insert Extended operation has the same effect in terms of logical currency, but can have a different effect in terms of physical currency. For example, executing a Get

Next (6) operation after either procedure produces the same result, but executing a Step Next (24) might return different records.

Open (0)

The Open operation makes a file available for access. To access a file, your application must first perform an Open operation. The file does not have to reside in the current directory as long as you specify the full or relative pathname.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X		X	X	X	X
Returned		X				

Prerequisites

◆ The file to be opened must exist on an accessible logical disk drive.

◆ A file handle must be available for the file.

Procedure

1. Set the Operation Code to 0.

2. If the file has an owner, specify the owner name, terminated by a binary 0, in the Data Buffer parameter.

3. Specify the length of the owner name, including the binary 0, in the Data Buffer Length parameter.

4. Place the pathname of the file to open in the Key Buffer parameter. Terminate the pathname with a blank or binary zero. The pathname can be up to 80 characters long, including the volume name and the terminator.

For more information about path names Btrieve supports, refer to *Getting Started*.

Note ———————————————————————————————

For your application to be compatible with Scalable SQL, limit the pathname to 64 characters.

5. In the Key Number parameter, specify one (or more) of the mode values listed in Table 2-15.

Details

In the Key Number parameter, you can add a -32 or -64 bias to the open mode values 0 through –4 to combine standard open modes with file sharing modes. For example –2 plus –64 (–66) tells the MicroKernel to open the file in Read-Only mode with MEFS enabled on that file.

Table 2-15   Open Modes

Mode	Description
0	Normal

Table 2-15   Open Modes *continued*

Mode	Description
−1	Accelerated To improve performance on specific files, you can open a file in Accelerated mode. (The 6.x MicroKernel accepted Accelerated mode opens, but interpreted them as Normal opens.) When you open a file in Accelerated mode, the MicroKernel does not perform transaction logging on the file. If you open a file in Accelerated mode with multi-engine file sharing (MEFS), local clients can access the file only if they also open it in Accelerated mode and with MEFS. No remote clients can access the file. If you open a file in Accelerated mode with single engine file sharing (SEFS), local clients can open the same file in SEFS mode combined with any open mode except Exclusive. For more information about MEFS and SEFS, refer to the *Btrieve Programmer's Guide.*
	NetWare developers: Opening a file in Accelerated mode cancels the effect of flagging a file as transactional. (On other platforms, the MicroKernel ignores the file's NetWare TTS flag.)
−2	Read-Only When you open a file in Read-Only mode, you can only read the file; you cannot perform updates. This mode allows you to open a file with corrupt data that the MicroKernel cannot automatically recover. If the data in the file's indexes has been corrupted, you can retrieve the records by opening the file in Read-Only mode and then using the Step Next (24) operation.
−3	Verify This mode is ignored. If you specify this mode, the MicroKernel opens the file in Normal mode. In previous versions of the MicroKernel, Verify mode verified that the data written to disk was correct.

Table 2-15   Open Modes *continued*

Mode	Description
−4	Exclusive Exclusive mode gives an application exclusive access to a file. No other application can open that file until the application that has exclusive access to the file closes it.
−32	Single Engine File Sharing Bias Use this open mode bias when you want to open a file in SEFS mode and the default file sharing mode for the drive is MEFS. **Note:** This bias works only if the MicroKernel's Enable Sharing Bias configuration option is turned on. If the Enable Sharing Bias option is turned off, this bias has no effect.
−64	Multi-Engine File Sharing Bias Use this open mode bias when you want to open a file in MEFS mode and the default file sharing mode for the drive is SEFS. **Note:** This bias works only if the MicroKernel's Enable Sharing Bias configuration option is turned on. If the Enable Sharing Bias option is turned off, this bias has no effect. For server MicroKernels, this mode maps to SEFS mode.

The MicroKernel allows a maximum of 250 open files, but you might be unable to open that many files due to limitations on system resources.

A file is opened only once by the MicroKernel. (The MicroKernel recognizes and handles the situation in which more than one client at a time opens a file, or a single client has more than one Position Block in the file.) When you open an extended file, the MicroKernel uses a single handle, and opens the base file and all extension files.

> **Note** ──
>
> When the NetWare server MicroKernel opens an extended file, it enforces NetWare
> security on the base file, but not on the extension files. In the rare event that a user has
> NetWare rights to a base file, but not the extension files, NetWare security can be
> violated. For workstation engines, NetWare does enforce security; therefore, such a
> user would not have access to the extension files.

Result

If the Open operation is successful, the MicroKernel assigns a file handle to the file,
reserves the Position Block passed on the Open call for the newly opened file, and makes
the file available for access.

If the Open operation is unsuccessful, the MicroKernel returns one of the following status
codes:

2	The application encountered an I/O error.
11	The specified filename is invalid.
12	The MicroKernel cannot find the specified file.
20	The MicroKernel or Btrieve Requester is inactive.
46	Access to the requested file is denied.
84	The record or page is locked.
85	The file is locked.
86	The file table is full.
87	The handle table is full.
88	The application encountered an incompatible mode error.

The following tables show the possible combinations for open modes involving local clients using SEFS, local clients using MEFS, and remote clients (implicitly using MEFS).

Table 2-16 shows open modes involving local clients using SEFS.

Table 2-16   Open Mode Combinations for Local Clients Using SEFS

Open Mode for Local Client 1	Open Mode for Local Client 2	Result
Normal	Normal	Successful
	Read-Only	Successful
	Exclusive	Status Code 88
	Accelerated	Successful
Read-Only	Normal	Successful
	Read-Only	Successful
	Exclusive	Status Code 88
	Accelerated	Successful
Exclusive	Normal	Status Code 88
	Read-Only	Status Code 88
	Exclusive	Status Code 88
	Accelerated	Status Code 88
Accelerated	Normal	Successful
	Read-Only	Successful
	Exclusive	Status Code 88
	Accelerated	Successful

Table 2-17 shows open modes involving local clients using MEFS.

Table 2-17   Open Mode Combinations for Local Clients Using MEFS

Open Mode for Local Client 1	Open Mode for Local Client 2	Result
Normal	Normal	Successful
	Read-Only	Successful
	Exclusive	Status Code 88
	Accelerated	Status Code 88
Read-Only	Normal	Successful
	Read-Only	Successful
	Exclusive	Status Code 88
	Accelerated	Status Code 88
Exclusive	Normal	Status Code 88
	Read-Only	Status Code 88
	Exclusive	Status Code 88
	Accelerated	Status Code 88
Accelerated	Normal	Status Code 88
	Read-Only	Status Code 88
	Exclusive	Status Code 88
	Accelerated	Successful

Table 2-18 shows open modes involving remote clients, which implies the use of multi-engine file sharing.

Table 2-18   Open Mode Combinations for Remote Clients

Open Mode for Remote Client 1	Open Mode for Remote Client 2	Result
Normal	Normal	Successful
	Read-Only	Successful
	Exclusive	Status Code 88
	Accelerated	Status Code 88
Read-Only	Normal	Successful
	Read-Only	Successful
	Exclusive	Status Code 88
	Accelerated	Status Code 88
Exclusive	Normal	Status Code 88
	Read-Only	Status Code 88
	Exclusive	Status Code 88
	Accelerated	Status Code 88
Accelerated	Normal	Status Code 88
	Read-Only	Status Code 88
	Exclusive	Status Code 88
	Accelerated	Status Code 88

Positioning

An Open operation does not establish any positioning except that the physical next record becomes the first physical record of the file.

Reset (28)

The Reset operation releases all resources held by a client. This operation aborts any transactions the client has pending, releases all locks, and closes all open files for the client.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X				X	X
Returned						

Prerequisites

Your application can issue a Reset operation at any time after the MicroKernel or Requester is loaded, as long as the client issuing the Reset call has established a connection with the MicroKernel—for example, by opening a file or by requesting the status of a file using a Btrieve utility.

Procedure

1. Set the Operation Code to 28.
2. Set the Key Number and Key Buffer to 0.

Result

If the Reset operation is successful, the MicroKernel performs the following actions for the specified client, window, or session:

1. Aborts any active transactions.
2. Releases all locks held.
3. Closes all open files.

If the Reset operation is unsuccessful, the MicroKernel returns a nonzero status code.

Positioning

The Reset operation destroys all currencies because it closes any open files.

Set Directory (17)

The Set Directory operation sets the current directory to a specified pathname.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X				X	
Returned						

Prerequisites

The target logical disk drive and directory must be accessible.

Procedure

1. Set the Operation Code to 17.

2. Store the logical disk drive and directory path, terminated by a binary 0, in the Key Buffer. If you omit the drive name, the MicroKernel uses the default drive. If you do not specify the complete path for the directory, the MicroKernel appends the directory path specified in the Key Buffer to the current directory.

   For more information about path names Btrieve supports, refer to *Getting Started*.

Result

If the Set Directory operation is successful, the MicroKernel makes the directory specified in the Key Buffer the current directory.

If the Set Directory operation is unsuccessful, the MicroKernel leaves the current directory unchanged and returns a nonzero status code.

Positioning

The Set Directory operation has no effect on positioning.

Set Owner (29)

The Set Owner operation assigns an owner name to a file, so that users who do not provide the name cannot access or modify the file. If an owner name has been set for a file, users or applications must specify the owner name each time they open the file. You can specify that an owner name be required for any access or just for update privileges.

When you assign an owner name, you can also direct the MicroKernel to encrypt the file's data on the disk. If you specify data encryption, the MicroKernel encrypts all the data during the Set Owner operation. Therefore, the longer the file, the longer Set Owner takes to complete.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X	X	X
Returned		X				

Prerequisites

- ◆ The file must be open.
- ◆ No transactions can be active.
- ◆ The file cannot already have an owner name.

Procedure

1. Set the Operation Code to 29.
2. Pass the Position Block that identifies the file to protect.

3. Store the owner name in both the Data Buffer and the Key Buffer. The MicroKernel requires that the name be in both buffers to avoid accidentally specifying an incorrect value. The owner name can be up to eight characters long and must end with a binary 0.

4. Specify the Data Buffer Length.

5. Set the Key Number to an integer that specifies the type of access restrictions and encryption for the file. (See Table 2-19.)

Details

Once you specify an owner name, it remains in effect until you issue a Clear Owner operation.

The following table lists the access restriction codes you can specify for the Key Number.

Table 2-19 Access and Encryption Codes

Code	Description
0	Requires an owner name for any access mode (no data encryption).
1	Permits read-only access without an owner name (no data encryption).
2	Requires an owner name for any access mode (with data encryption).
3	Permits read-only access without an owner name (with data encryption).

Result

If the Set Owner operation is successful, the MicroKernel prevents future operations from accessing or modifying the file unless those operations specify the correct owner name. The only exception is if read-only access is allowed without an owner name. In addition,

if the Set Owner operation is successful, the MicroKernel encrypts the data in the file (if encryption is specified).

Encryption occurs immediately; the MicroKernel has control until the entire file is encrypted, and the larger the file, the longer the encryption process takes. Reading data from an encrypted file is slower than reading data from an unencrypted file. The MicroKernel decrypts a page when it loads the page from the disk, then encrypts the page when it writes to the disk again. If you have a small cache or use a relatively large amount of modification operations, the MicroKernel must execute the encryption routine more frequently. To enhance performance, the MicroKernel performs logical operations (such as SHIFT and XOR) on the bytes of the page using the encrypted owner name as a key.

If the Set Owner operation is unsuccessful, the MicroKernel returns one of the following status codes:

41	The MicroKernel does not allow the attempted operation.
50	The file owner is already set.
51	The owner name is invalid.

Positioning

The Set Owner operation has no effect on positioning.

Stat (15)

The Stat operation retrieves the defined characteristics of a file and statistics about the file's contents, such as the number of records in the file, the number of unique key values stored for each index in the file, and the number of unused pages in the file.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X	X	X
Returned			X	X	X	

Prerequisites

The file must be open.

Procedure

1. Set the Operation Code to 15.
2. Pass the Position Block for the file.
3. Indicate a Data Buffer to hold the statistics defined for the file.
4. Specify the Data Buffer Length, which must be long enough to hold the file statistics. (For more information, see Table 2-20 and Table 2-21.)
5. Specify a Key Buffer that is at least 255 characters long.

6. Set the Key Number as follows:

- ◆ Specify 0 to exclude file version and unused duplicate pointers information. Parse the returned Data Buffer as shown in <u>Table 2-20</u>.

- ◆ Specify –1 (0xFF) to include file version and unused duplicate pointers information. Parse the returned Data Buffer as shown in <u>Table 2-21</u>.

Details

The MicroKernel returns information about all keys in the file, including those added since file creation. The key information includes any applicable ACS definitions. You must account for this extra information in the Data Buffer. Specifically, do not use the same Data Buffer here that you used for the Create (0) operation.

Because the MicroKernel allows up to 119 keys and multiple ACSs in a file, the longest possible Data Buffer is 33,455 bytes (that is, 16 + (11 * 16) + (119 * 265)). However, you probably do not need such a large data buffer. In fact, you may prefer to specify a smaller Data Buffer if you want only certain information. For example, you could set the Data Buffer Length to 1920 bytes (that is, 16 + (16 * 119)). In effect, such a setting returns all key information but not necessarily all the ACSs. If your application does not need information about the ACSs, you might prefer this method.

If you specify a value of 0 in the Key Number parameter, the MicroKernel returns Stat information as shown in the following table.

Table 2-20   Data Buffer Excluding File Version Information

Element	Description	Length (in Bytes)
File Specification	Record Length	2
	Page Size	2
	Number of Indexes	2
	Number of Records	4
	File Flags	2
	Reserved Word	2
	Unused Pages	2
Key Specification (repeated for each segment)	Key Position	2
	Key Length	2
	Key Flags	2
	Number of Unique Key Values	4
	Extended Data Type	1
	Null Value	1
	Reserved	2
	Key Number	1
	ACS Number	1

Table 2-20   Data Buffer Excluding File Version Information *continued*

Element	Description	Length (in Bytes)
ACS Number 0	ACS	265
. . .	. . .	. . .
ACS Number x	ACS	265

If you specify a value of –1 in the Key Number parameter, the MicroKernel returns Stat information as shown in the following table.

Table 2-21   Data Buffer Including File Version Information

Element	Description	Length (in Bytes)
File Specification	Record Length	2
	Page Size	2
	Number of Indexes	1
	File Version Number	1
	Number of Records	4
	File Flags	2
	Number of Unused Duplicate Pointers	1
	Reserved Byte	1
	Unused Pages	2

Table 2-21   Data Buffer Including File Version Information *continued*

Element	Description	Length (in Bytes)
Key Specifications (repeated for each key segment)	Key Position	2
	Key Length	2
	Key Flags	2
	Number of Unique Key Values	4
	Extended Data Type	1
	Null Value	1
	Reserved	2
	Key Number	1
	ACS Number	1
ACS Number 0	ACS	265
. . .	. . .	. . .
ACS Number x	ACS	265

File Specifications

The File Specification fields in the returned Data Buffer are the same as those described for "Create (14)", with the following exceptions:

- ◆ In the File Specification area:

  - ◆ If the Data Buffer includes file version information, the Number of Indexes is 1 byte long and is followed by a 1-byte File Version Number. Do not translate the File Version Number value to decimal. A value of 0x70 indicates that the file is a 7.0 file; a value of 0x60 indicates that the file is 6.*x*, and so on. When creating a file, the MicroKernel assigns a version number according to the attributes defined for the file.

- The Number of Records is a 4-byte value representing the number of records in the file.

- In the File Flags word, Bits 9 and 12 have the following meaning:

Bit 9 = 1 and Bit 12 = 0	File was created with system data. (This does not necessarily mean that the system-defined log key is currently in use; it may have been dropped.)
Bit 9 = 1 and Bit 12 = 1	File was created without system data.

  Stat does not indicate whether system data was included by default or explicitly.

- If the Data Buffer includes file version information, a 1-byte Number of Unused Duplicate Pointers follows the File Flags field and indicates how many unused duplicate pointers remain in the file.

- The reserved areas are allocated even though the MicroKernel ignores them on a Stat operation.

Key Specifications

The Key Specification fields in the returned Data Buffer are the same as those described for "Create (14)", except that a 4-byte Number of Unique Key Values follows the Key Flags field and indicates the number of records that have a unique, non-duplicated value for the specified key.

ACSs

The ACS definitions in the returned Data Buffer are the same as those described for "Create (14)".

Result

If the Stat operation is successful, the MicroKernel returns the file and key characteristics to the Data Buffer and the length of the Data Buffer in the Data Buffer Length. If the file is an extended file, the MicroKernel returns the filename of the first extension file in the Key Buffer. If the filename of the first extension file is longer than 63 bytes, the MicroKernel truncates the filename. If the file is not an extended file, the MicroKernel initializes the first byte of the Key Buffer to 0. (You can use the Stat Extended operation to retrieve statistics regarding extended files.)

If the Stat operation is unsuccessful, the MicroKernel returns one of the following status codes:

3	The file is not open.
22	The data buffer parameter is too short.

Positioning

The Stat operation has no effect on positioning.

Stat Extended (65)

The Stat Extended operation has two subfunctions. One subfunction returns the filenames and paths of an extended file's components: the base file and extension files. The other subfunction reports whether a file is using a system-defined log key, and by implication, whether the file is transaction durable.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X		X
Returned			X	X		

Prerequisites

The file must be open.

Procedure

1. Set the Operation Code to 65.

2. Pass the Position Block for the file.

3. Store the extended stat structure in the Data Buffer. See the "Details" section for more information about the extended stat structure.

4. Specify the Data Buffer Length.

5. Set the Key Number to 0.

Details

To receive information about an extended file's components, you must create an extended files descriptor in the Data Buffer, as follows.

Table 2-22   Extended Files Descriptor

Element	Length (in Bytes)	Description
Signature	4	Type of extended stat call. Specify 0x74537845. (This is equivalent to ASCII *ExSt.*)
Subfunction	4	Type of extended stat call. Specify 0x00000001.
Namespace	4	File naming convention. Specify 0x00000000.
Max Files	4	Maximum number of filenames to return. You can set this value higher than the number of extension files composing the extended file. (An extended file can contain up to 32 extension files.)
First File Sequence	4	Sequence number of the first filename to return. Specify 0 to begin with the base file, 1 to begin with the first extension file, and so on. If you specify a number higher than the number of extension files, the MicroKernel returns Status Code 0 and no filenames.

To receive information about a file's use of system data, you must create a system data descriptor in the Data Buffer, as follows.

Table 2-23   System Data Descriptor

Element	Length (in Bytes)	Description
Signature	4	Type of extended stat call. Specify 0x74537845. (This is equivalent to ASCII *ExSt.*)

Table 2-23   System Data Descriptor

Element	Length (in Bytes)	Description
Subfunction	4	Type of extended stat call. Specify 0x00000002.

Result

If the Stat Extended operation is successful, the MicroKernel returns the file's statistics in the Data Buffer and sets the Data Buffer Length parameter to the number of bytes returned.

For the extended files subfunction, the MicroKernel returns an extended files structure in the Data Buffer, as follows.

Table 2-24   Extended Files Return Buffer

Element	Length (in Bytes)	Description
Number of Files	4	Number of operating system files that comprise the extended file.
Number of Extensions	4	Number of extension files returned.
Filename Portion (Repeated for each filename returned)		
Length of Filename	4	Length of the extension filename.
Filename	*n*	Extension filename.

For the system data subfunction, the MicroKernel returns a system data structure in the Data Buffer, as follows.

Table 2-25   System Data Return Buffer

Element	Length (in Bytes)	Description
Has System Data	1	Indicates whether the file's records contain a system-defined log key (also called system data). 1 = Yes and 0 = No.
Has Log Key	1	Indicates whether the system-defined log key is currently being used, as opposed to being dropped. 1 = Yes and 0 = No.
Is Loggable	1	Indicates whether the file has a unique key that can be used to implement transaction durability. This key can be either a user-defined unique key or a system-defined log key. 1 = Yes and 0 = No.
Log Key Number	1	Key number that the MicroKernel is currently using as the transaction log key. If the system-defined log key is being used as the transaction log key, this value is 125.
Size of System Data	2	Size of the system-defined log key, which is 8.
System Data Version	2	The constant 700 (0x2BC).

If the Stat Extended operation is unsuccessful, the MicroKernel returns one of the following status codes:

   3   The file is not open.

 06   The key number parameter is invalid.

 22   The data buffer parameter is too short.

 62   The descriptor is incorrect.

Step First (33)

The Step First operation retrieves the first physical record of the file. The MicroKernel does not use a key path to retrieve the record.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X		X		
Returned		X	X	X		

Prerequisites

The file must be open.

Procedure

1. Set the Operation Code to 33. Optionally, you can include a lock bias:

   - 100—Single wait record lock.

   - 200—Single no-wait record lock.

   - 300—Multiple wait record lock.

   - 400—Multiple no-wait record lock.

   For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Set the Data Buffer Length to a value greater than or equal to the length of the record to retrieve.

Result

If the Step First operation is successful, the MicroKernel returns the file's first physical record in the Data Buffer and sets the Data Buffer Length parameter to the number of bytes returned.

If the Step First operation is unsuccessful, the MicroKernel returns one of the following status codes:

3	The file is not open.
9	The operation encountered the end-of-file.
22	The data buffer parameter is too short.

Positioning

The Step First operation destroys logical currency. Step First sets the physical currency using the retrieved record as the physical current record. The previous physical position points beyond the beginning of the file.

Step Last (34)

The Step Last operation retrieves the last physical record of the file. The MicroKernel does not use a key path to retrieve the record.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X		X		
Returned		X	X	X		

Prerequisites

The file must be open.

Procedure

1. Set the Operation Code to 34. Optionally, you can include a lock bias:

    *   100—Single wait record lock.

    *   200—Single no-wait record lock.

    *   300—Multiple wait record lock.

    *   400—Multiple no-wait record lock.

    For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Set the Data Buffer Length to a value greater than or equal to the length of the record to retrieve.

Result

If the Step Last operation is successful, the MicroKernel returns the file's last physical record in the Data Buffer and sets the Data Buffer Length parameter to the number of bytes returned.

If the Step Last operation is unsuccessful, the MicroKernel may return one of the following status codes:

   3   The file is not open.

   9   The operation encountered the end-of-file. (when the file is empty)

  22   The data buffer parameter is too short.

Positioning

The Step Last operation destroys logical currency. Step Last sets the physical currency using the retrieved record as the current physical record. The next physical position points beyond the end of the file.

Step Next (24)

The Step Next operation retrieves the record to which the next physical position points. The MicroKernel does not use a key path to retrieve the record.

A Step Next operation issued immediately after any Get or Step operation returns the record physically following the record retrieved by the previous operation.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X		X		
Returned		X	X	X		

Prerequisites

The file must be open.

Procedure

1. Set the Operation Code to 24. Optionally, you can include a lock bias:
   - 100—Single wait record lock.
   - 200—Single no-wait record lock.
   - 300—Multiple wait record lock.
   - 400—Multiple no-wait record lock.

   For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Set the Data Buffer Length to a value greater than or equal to the length of the record to retrieve.

Result

If the Step Next operation is successful, the MicroKernel returns the file's next physical record in the Data Buffer and sets the Data Buffer Length parameter to the number of bytes returned.

If the Step Next operation is unsuccessful, the MicroKernel returns one of the following status codes:

3	The file is not open.
9	The operation encountered the end-of-file.
22	The data buffer parameter is too short.

Positioning

The Step Next operation does not establish logical currency. Step Next sets the physical currency using the retrieved record as the physical current record.

If a Step Next operation is issued immediately following a Delete operation (4), Step Next returns the record that was established as the next physical record by the operation preceding the Delete.

If a Step Next operation is issued immediately after an Open operation (0), Step Next returns the first record in the file.

Step Next Extended (38)

The Step Next Extended operation examines one or more records, starting at the next physical position and proceeding toward the end of the file. It checks to see if the examined record or records satisfy a filtering condition, and it retrieves the ones that do. The filtering condition is a logic expression and is not limited to key fields only.

Step Next Extended can also extract specified fields from existing records and return a new set of records that contain only the extracted fields.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X		
Returned		X	X	X		

Prerequisites

- ◆ The file must be open.

- ◆ You must have established a next physical position. (For example, a Step Next Extended operation cannot follow a Delete operation.)

Procedure

1. Set the Operation Code to 38. Optionally, you can include a lock bias:
   - ◆ 100—Single wait record lock.
   - ◆ 200—Single no-wait record lock.
   - ◆ 300—Multiple wait record lock.
   - ◆ 400—Multiple no-wait record lock.

For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Specify a Data Buffer large enough to accommodate either the input data buffer or the returned data buffer, whichever is larger. Initialize the Data Buffer according to the structure shown in Table 2-12.

4. Specify the Data Buffer Length from the preceding step.

   The MicroKernel sets up a buffer as a workspace for extended operations. You configure the size of this buffer using the Extended Operation Buffer Size option. The sum of the Data Buffer structure, plus the longest record to be retrieved, plus 355 bytes of requester overhead, cannot exceed the configured buffer size. (Requester overhead is not applicable in DOS workstation engines.)

Details

The Step Next Extended operation shares the same "Details" as the Get Next Extended operation. Refer to "Details" for more information.

Result

If the Step Next Extended operation is successful, the MicroKernel returns one or more fields from one or more records to the Data Buffer (as shown in Table 2-13). The MicroKernel also sets the Data Buffer Length parameter to the number of bytes it returned to the Data Buffer.

If the Step Next Extended operation is unsuccessful, the MicroKernel returns one of the following status codes:

3	The file is not open.
9	The operation encountered the end-of-file.
22	The data buffer parameter is too short.

60	The specified reject count has been reached.
61	The work space is too small.
62	The descriptor is incorrect.
64	The filter limit has been reached.
65	The field offset is incorrect.
82	The MicroKernel lost positioning.
134	The MicroKernel cannot read the International Sorting Rule.
135	The specified International Sort Rule table is corrupt or otherwise invalid.
136	The MicroKernel cannot find the specified Alternate Collating Sequence in the file.

It is possible for the MicroKernel to return a nonzero status code and also return valid data in the Data Buffer. In this case, the last record returned may be incomplete. If the Data Buffer Length parameter returned is greater than 0, check the Data Buffer for extracted data.

If a field can only be partially filled because the record is too short, then the MicroKernel returns what it can of the record to and including the partial field. If the partial field is the last field to be extracted, then the MicroKernel continues the operation. Otherwise, the MicroKernel aborts the operation and returns a Status Code 22.

For example, consider a Step Next Extended operation that retrieves three fields from two variable-length records. The first record is 55 bytes long, and the second is 50 bytes. The fields to be retrieved are defined as follows:

◆ Field 1 begins at offset 2 and is 2 bytes long.

◆ Field 2 begins at offset 45 and is 10 bytes long.

◆ Field 3 begins at offset 6 and is 2 bytes long.

When the MicroKernel performs the Step Next Extended operation, it returns the first record without any problem. However, when attempting to extract 10 bytes from field 2 of the second record, the MicroKernel finds that only 5 bytes are available (between offset 45 and the end of the record, at offset 49). At this point, the MicroKernel does not pad the missing 5 bytes of field 2, and thus cannot extract field 3. Instead, the MicroKernel returns Status Code 22 and places all of field 1 and first 5 bytes of field 2 in the return Data Buffer.

Positioning

The Step Next Extended operation does not establish any logical currency, but the last record examined (not necessarily retrieved) becomes the current physical record. This record can be either a record that satisfies the filtering condition and is retrieved, or a record that does not satisfy the filtering condition and is rejected.

Note

The MicroKernel does not allow Delete or Update operations after a Step Next Extended operation. Because the current record is the last record examined, there is no way to ensure that your application would delete or update the intended record.

Step Previous (35)

The Step Previous operation retrieves the record to which the previous physical position points. The MicroKernel does not use an index path to retrieve a record for a Step Previous operation.

A Step Previous operation performed immediately after any Get or Step operation returns the record physically preceding the record that the previous operation retrieves.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X		X		
Returned		X	X	X		

Prerequisites

- ◆ The file must be open.

- ◆ You must have an established previous physical position. (For example, a Step Previous cannot follow a Delete operation.)

Procedure

1. Set the Operation Code to 35. Optionally, you can include a lock bias:

   - ◆ 100—Single wait record lock.

   - ◆ 200—Single no-wait record lock.

   - ◆ 300—Multiple wait record lock.

   - ◆ 400—Multiple no-wait record lock.

For more information about locking, refer to the *Btrieve Programmer's Guide.*

2. Pass the Position Block for the file.

3. Set the Data Buffer Length to a value greater than or equal to the length of the record to retrieve.

Result

If the operation is successful, the MicroKernel returns the previous physical record in the Data Buffer and sets the Data Buffer Length parameter to the number of bytes returned.

If the operation is unsuccessful, the MicroKernel returns one of the following status codes:

3	The file is not open.
9	The operation encountered the end-of-file. (at the beginning of the file)
22	The data buffer parameter is too short.

Positioning

The Step Previous operation does not establish logical currency. Step Previous sets the physical currency using the retrieved record as the physical current record.

Step Previous Extended (39)

The Step Previous Extended operation examines one or more records, starting at the previous physical position and proceeding toward the beginning of the file. It checks to see if the examined record or records satisfy a filtering condition, and it retrieves the ones that do. The filtering condition is a logic expression and is not limited to key fields only.

Step Previous Extended can also extract specified fields from existing records and return a new set of records that contain only the extracted fields.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X		
Returned		X	X	X		

Prerequisites

◆ The file must be open.

◆ You must have established a previous physical position. (For example, a Step Previous Extended operation cannot follow a Delete operation.)

Procedure

1. Set the Operation Code to 39. Optionally, you can include a lock bias:
   - 100—Single wait record lock.
   - 200—Single no-wait record lock.
   - 300—Multiple wait record lock.
   - 400—Multiple no-wait record lock.

   For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file.

3. Specify a Data Buffer large enough to accommodate either the input data buffer or the returned data buffer. Initialize the Data Buffer according to the structure shown in Table 2-12.

4. Specify the Data Buffer Length from the preceding step.

The MicroKernel sets up a buffer as a workspace for extended operations. You configure the size of this buffer using the Extended Operation Buffer Size option. The sum of the Data Buffer structure, plus the longest record to be retrieved, plus 355 bytes of requester overhead, cannot exceed the configured buffer size. (Requester overhead is not applicable in DOS workstation engines.)

Details

This operation uses the same input and returned Data Buffers as the Get Next Extended operation. Refer to "Details" for more information.

Result

This operation returns the same results as the Step Next Extended operation. Refer to "Result" for more information.

Positioning

The Step Previous Extended operation does not establish logical currency, but the last record examined (not necessarily retrieved) becomes the current physical record. This record can be either a record that satisfies the filtering condition and is retrieved, or a record that does not satisfy the filtering condition and is rejected.

Note

The MicroKernel does not allow Delete or Update operations after a Step Previous Extended operation. Because the current record is the last record examined, there is no way to ensure that your application would delete or update the intended record.

Stop (25)

The Stop operation performs a number of termination routines for the client, such as releasing all locks and closing all open files associated with that client.

> **Note**
>
> *OS/2 developers*: The Stop operation behaves the same as "Reset (28)".

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X					
Returned						

Procedure

Set the Operation Code to 25.

Result

If the Stop operation is successful, the MicroKernel performs the following actions:

1. Aborts any active transactions.
2. Releases all locks held by the client.
3. Closes all files open for the client.
4. If no other clients (other applications registered with the MicroKernel) exist and depending on the MicroKernel configuration, the MicroKernel may terminate itself and free a number of resources.

If the Stop operation is unsuccessful, the MicroKernel returns a nonzero status code. The most common nonzero Status Code is 20 (Record Manager Inactive). This status occurs because the MicroKernel or the Requester is not loaded.

Positioning

The Stop operation destroys all currencies because it closes any open files.

Unlock (27)

The Unlock operation unlocks one or more records that have been locked explicitly (that is, the records were locked using a lock bias of +100, +200, +300, or +400). The Unlock operation releases locks held by the specified position block; therefore, if you have the same file opened more than once, you must issue an Unlock for each position block before the record is completely unlocked. Similarly, each client that holds a lock on records in the file must issue an Unlock before the record is completely unlocked.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X		X
Returned						

Prerequisites

You must have at least one record lock.

Procedure

To unlock a single-record lock, follow these steps:

1. Set the Operation Code to 27.

2. Pass the Position Block for the file that contains the locked record.

3. Set the Key Number to a non-negative value.

To unlock a record locked by a multiple-record lock, first retrieve the 4-byte position of the record to unlock by issuing a Get Position operation (22) for that record. Then, issue the Unlock operation as follows:

1. Set the Operation Code to 27.

2. Pass the Position Block for the file that contains the locked record.

3. Store (in the Data Buffer) the 4-byte position that the MicroKernel returns.

4. Set the Data Buffer Length to 4.

5. Set the Key Number parameter to –1.

To unlock all the multiple record locks on a file, follow these steps:

1. Set the Operation Code to 27.

2. Pass the Position Block for the file that contains the multiple locks.

3. Set the Key Number parameter to –2.

Result

If the Unlock operation is successful, the MicroKernel releases all the locks that the operation specified.

If the Unlock operation is unsuccessful, the MicroKernel returns a nonzero status code—most likely, Status Code 81.

Positioning

The Unlock operation has no effect on positioning.

Update (3)

The Update operation changes the information in an existing record.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X		X
Returned		X			X	

Note ——————————————————————————

When using the no-currency-change (NCC) option, the Update operation does not update the value of the Key Buffer parameter; it does not return any information in that parameter.

Prerequisites

◆ The file must be open.

◆ You must have established physical currency in the file. (Although an Extended Get, Extended Step, or Get Key operation establishes the required position, these operations cannot be followed by an Update.)

◆ In order to update a record while inside a transaction, that record must have been retrieved while inside that transaction, and not before.

Procedure

1. Set the Operation Code to 3. Optionally, you can include a lock bias:

    *   100—Single wait record lock.

    *   200—Single no-wait record lock.

    *   300—Multiple wait record lock.

    *   400—Multiple no-wait record lock.

    For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file containing the record.

3. Store the updated data record in the Data Buffer.

4. Set the Data Buffer Length to the length of the updated record.

5. Set the Key Number used for retrieving the record. To use the NCC option, specify –1 (0xFF) for the Key Number. To use the system-defined log key (also called system data), specify 125.

    When performing a non-NCC Update operation immediately following a Get operation, pass the Key Number exactly as the MicroKernel returned it on the Get operation; otherwise, the MicroKernel updates the record successfully but returns Status Code 7 on the first Get operation performed after the update.

Result

If the Update operation is successful, the MicroKernel updates the record stored in the file with the new value in the Data Buffer, adjusts the indexes to reflect any change in the key values, and returns the value of the specified key in the Key Buffer. An NCC Update operation does not update the value of the Key Buffer parameter.

If the application holds a single-record lock on the record to be updated, the MicroKernel releases the lock. However, a multiple-record lock is never released by an Update operation.

If the Update operation is unsuccessful, the MicroKernel returns one of the following status codes:

5	The record has a key field containing a duplicate key value.
8	The current positioning is invalid.
10	The key field is not modifiable.
22	The data buffer parameter is too short.
80	The MicroKernel encountered a record-level conflict.
83	The MicroKernel attempted to update or delete a record that was read outside the transaction.

Positioning

The Update operation and the NCC Update operation do not affect physical currency.

An Update operation that does not use the NCC option can affect logical currency if the value of the updated key repositions the record in the index. For example, suppose an INTEGER key's logical current record has a value of 1. For that same key, the logical next record has a value of 2. If you update 1 to 4, you no longer have the same logical next record. In this example, after the Update operation, the logical next record has a value that is greater than 4.

An NCC Update operation does not affect logical currency. This means that an application, having performed an NCC Update operation, has the same logical position in the file as it had prior to the Update operation. In such a situation, operations that follow

an NCC Update—such as Get Next (6), Get Next Extended (36), Get Previous (7), and Get Previous Extended (37)—return values based on the application's logical currency prior to the NCC Update.

Note

The MicroKernel does not return any information in the Key Buffer parameter as the result of an NCC Update operation. Therefore, an application that must maintain the logical currency must not change the value of the Key Buffer following the NCC Update operation. Otherwise, the next Get operation has unpredictable results.

Update Chunk (53)

The Update Chunk operation can change the information in one or more portions of a record (each portion being a chunk). It can also append information to an existing record (thereby lengthening the record), or truncate an existing record at a specified offset.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X	X	X	X		X
Returned		X			X	

Prerequisites

◆ The file must be open.

◆ You must have an established current physical or logical record in the file.

◆ To update any chunk within a record while inside a transaction, the record must have been retrieved while inside that transaction and not before.

> **Note**
>
> Although an extended operation or a Get Key operation (+50) establishes the required position, you cannot issue an Update Chunk operation immediately after these operations, because they do not return a single record.

Procedure

1. Set the Operation Code to 53. Optionally, you can include a lock bias:

    - 100—Single wait record lock.

    - 200—Single no-wait record lock.

    - 300—Multiple wait record lock.

    - 400—Multiple no-wait record lock.

    For more information about locking, refer to the *Btrieve Programmer's Guide*.

2. Pass the Position Block for the file containing the record.

3. Specify a Data Buffer, as described in "Details".

4. Set the Data Buffer Length to a value greater than or equal to the number of bytes your application has placed in the Data Buffer. See the "Details" section for more information about calculating the Data Buffer Length.

5. Set the Key Number used for retrieving the record in the Key Number parameter. To use the system-defined log key (also called system data), specify 125.

Details

Use one of the following chunk descriptors in the Data Buffer:

- Random Chunk Descriptor—To update a single chunk per operation, or to update more than one chunk in a single operation when the chunks are spaced randomly throughout the record.

- Rectangle Chunk Descriptor—To update many chunks in an operation, when each chunk is the same length and chunks are spaced equidistantly in the record.

- Truncate Chunk Descriptor—To truncate a record at a specified offset.

Random Chunk Descriptor Structure

The following example shows a record with three randomly spaced chunks (shaded areas): chunk 0 (bytes 0x12 through 0x16), chunk 1 (bytes 0x2A through 0x31), and chunk 2 (bytes 0x41 through 0x4E).

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

To define a random chunk descriptor, your application must create a structure in the Data Buffer, based on the following table.

Table 2-26   Random Chunk Descriptor Structure

Element	Length (in Bytes)	Description
Subfunction	4	Type of chunk descriptor; one of the following: ◆ 0x80000000 (Direct random chunk descriptor)—Updates chunks stored directly in the Data Buffer. The data for updating the first chunk is stored in the Data Buffer immediately after the last chunk definition (Chunk *n*), the data for the second chunk immediately follows the first, and so on. ◆ 0x80000001 (Indirect random chunk descriptor)—Updates chunks from data at addresses specified by the Chunk Definitions.

Table 2-26   Random Chunk Descriptor Structure

Element	Length (in Bytes)	Description
NumChunks	4	Number of chunks to be updated. The value must be at least 1. Although no explicit maximum value exists, the chunk definitions must fit in the Data Buffer, which is limited in size as described in Table 2-9.
Chunk Definition (Repeat for each chunk)	12	Each Chunk Definition is a 4-byte Chunk Offset, followed by a 4-byte Chunk Length, followed by a 4-byte User Data, described as follows: ◆ Chunk Offset—Indicates where the chunk begins as an offset in bytes from the beginning of the record. The minimum value is 0, and the maximum value is the offset of the last byte in the record, plus 1. ◆ Chunk Length—Indicates how many bytes are in the chunk. The minimum value is 0, and the maximum value 65,535; however, the chunk definitions must fit in the Data Buffer, which is limited in size as described in Table 2-9. ◆ User Data—(Used only for indirect descriptors.) A 32-bit pointer to the actual chunk data. The format you should use depends on your operating system.[1] The MicroKernel ignores this element for direct chunk descriptor subfunctions.

1 For 16-bit applications, initialize User Data as a 16-bit offset and a 16-bit segment. User Data cannot address memory beyond the end of its segment. When Chunk Length is added to the offset portion of User Data, the result must be within the segment that User Data defines. By default, the MicroKernel does not check for violations of this rule and does not properly handle such violations.

The following table shows a sample direct random chunk descriptor structure.

Element	Sample Value	Length (in Bytes)
Subfunction	0x8000000	4
NumChunks	3	4

Element	Sample Value	Length (in Bytes)
Chunk 0		
Chunk Offset	0x12	4
Chunk Length	0x05	4
User Data	N/A	4
Chunk 1		
Chunk Offset	0x2A	4
Chunk Length	0x08	4
User Data	N/A	4
Chunk 2		
Chunk Offset	0x41	4
Chunk Length	0x0E	4
User Data	N/A	4
Data for Chunk 0	N/A	5
Data for Chunk 1	N/A	8
Data for Chunk 2	N/A	14

Rectangle Chunk Descriptor Structure

When chunks of the same length are spaced equidistantly throughout a record, you can describe all the chunks to update with a rectangle chunk descriptor. For example, consider the following diagram, which represents offset 0x00 through 0x4F in a record:

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

The record contains three chunks (shaded areas): chunk 0 (bytes 0x19 through 0x1C), chunk 1 (bytes 0x29 through 0x2C), and chunk 2 (bytes 0x39 through 0x3C). Each chunk is four bytes long, and a total of 16 (0x10) bytes, calculated from the beginning of each chunk, separates the chunks from one another.

You can update all three chunks using a single rectangle descriptor. To update rectangle chunks, you must create a structure in the Data Buffer based on Table 2-27.

Table 2-27   Rectangle Chunk Descriptor Structure

Element	Length (in Bytes)	Description
Subfunction	4	Type of chunk descriptor; one of the following: ◆ 0x80000002 (Direct rectangle chunk descriptor)—Updates chunks stored directly in the Data Buffer. The data for updating the first chunk is stored in the Data Buffer immediately after the last chunk definition (Chunk n), the data for the second chunk immediately follows the first, and so on. ◆ 0x80000003 (Indirect rectangle chunk descriptor)—Updates chunks from data at addresses specified by the Chunk Definitions.
Number of Rows	4	Number of chunks on which the rectangle chunk descriptor must operate. The minimum value is 1. No explicit maximum value exists.
Offset	4	Offset from the beginning of the record of the first byte to update. The minimum value is 0, and the maximum value is the offset of the last byte in the record, plus 1. If the record is viewed as a rectangle, this element refers to the offset of the first byte in the first row to be retrieved.
Bytes Per Row	4	Number of bytes in each chunk to be updated. The minimum value is 0, and the maximum value is 65,535; however, the chunk definitions must fit in the Data Buffer, which is limited in size as described in Table 2-9.
Distance Between Rows	4	Number of bytes between the beginning of each chunk.
User Data	4	(Used only with indirect descriptors.) A 32-bit pointer to the actual chunk data. The format you should use depends on your operating system.[1] The MicroKernel ignores this element for direct rectangle descriptors; however, you must still allocate the element and initialize it to 0.

Table 2-27   Rectangle Chunk Descriptor Structure

Element	Length (in Bytes)	Description
Application Distance Between Rows	4	(Used only with indirect rectangle descriptors.) Number of bytes between the beginning of chunks in the rectangle, as the rectangle is stored in your application's memory, at the address specified by User Data. The MicroKernel ignores this element for direct rectangle descriptors; however, you must still allocate the element and initialize it to 0.

1 For 16-bit applications, express User Data as a 16-bit offset followed by a 16-bit segment.

If the rectangle has the same number of bytes between rows when it is in memory as when it is stored as a record, set Application Distance Between Rows with the same value as Distance Between Rows. However, if the rectangle is arranged in your application's memory with either more or fewer bytes between rows, Application Distance Between Rows allows you to pass that information to the MicroKernel.

When you use an indirect rectangle descriptor, the MicroKernel uses both the User Data and the Application Distance Between Rows elements to determine the locations from which to read the data for the update. The MicroKernel reads data for the first row from offset 0 of User Data. The MicroKernel reads the second row's data from an address specified by User Data plus Application Distance Between Rows. The MicroKernel reads the third row's data from the address specified by User Data plus (Application Distance Between Rows * 2), and so on.

The following table shows a sample direct rectangle chunk descriptor structure.

Element Name	Sample Value	Length (in Bytes)
Subfunction	0x80000002	4

Element Name	Sample Value	Length (in Bytes)
Number of Rows	3	4
Offset	0x19	4
Bytes Per Row	0x04	4
Distance Between Rows	0x10	4
User Data	0	4
Application Distance Between Rows	0	4
Data (Row 0)	N/A	4
Data (Row 1)	N/A	4
Data (Row 2)	N/A	4

Truncate Descriptor Structure

The truncate descriptor allows you to truncate a record at a specified offset. To use this type of chunk descriptor, you must create a structure in the Data Buffer, based on the following table:

Table 2-28   Truncate Descriptor Structure

Element	Length (in Bytes)	Description
Subfunction	4	Type of chunk descriptor. Specify 0x80000004.
ChunkOffset	4	Byte offset into the record where truncation begins. That byte and every byte following it is eliminated. The minimum value is 4. The maximum value is the offset of the final byte in the record.

Next-in-Record Subfunction Bias

If you add a bias of 0x40000000 to any of the subfunctions previously listed, the MicroKernel calculates the subfunction's Offset element values based on your physical intrarecord currency (that is, your current physical position within the record). When you use the Next-in-Record subfunction, the MicroKernel ignores the Offset element in the chunk descriptor.

If you use this bias in combination with a random chunk descriptor and it updates more than one chunk in a single operation, the MicroKernel calculates the offset for all chunks (except the first) by adding the previous chunk's length to the previous chunk's offset. In other words, the next-in-record bias applies to all chunks in the operation.

Append Subfunction Bias

If you add a bias of 0x20000000 to the random chunk descriptor's subfunction or to the rectangle chunk descriptor's subfunction, the MicroKernel calculates the subfunction's Offset element value to be one byte beyond the end of the record.

> **Note**
>
> Do not use this bias with the Next-in-Record bias or the Truncate subfunction.

If you use this bias in combination with a random chunk descriptor and it updates more than one chunk in a single operation, the MicroKernel calculates the offset for all chunks (except the first chunk) based on the record's length after the MicroKernel appends the previous chunk.

Result

If the Update Chunk operation is successful, the MicroKernel updates the portions of the record identified as chunks in the chunk descriptor portion of the Data Buffer. The new

data for updating the chunks is contained either in the chunk descriptor itself (for direct chunk descriptor subfunctions) or in the memory address specified by the 32-bit pointer in each chunk's User Data element (for indirect chunk descriptor subfunctions). After the Update Chunk operation completes, the MicroKernel adjusts the key indexes to reflect any change in the key values, and, if necessary, updates the Key Buffer parameter.

In addition, if the application holds a single-record lock on the record to be updated, the MicroKernel releases the lock. However, a multiple-record lock is never released by an Update Chunk operation.

If the Update Chunk operation is unsuccessful, the MicroKernel returns one of the following status codes:

5	The record has a key field containing a duplicate key value.
8	The current positioning is invalid.
10	The key field is not modifiable.
22	The data buffer parameter is too short.
58	The compression buffer length is too short.
62	The descriptor is incorrect.
80	The MicroKernel encountered a record-level conflict.
83	The MicroKernel attempted to update or delete a record that was read outside the transaction.
97	The data buffer is too small.
103	The chunk offset is too big.
106	The MicroKernel cannot perform a Get Next Chunk operation.

Positioning

The Update Chunk operation does not change the physical currency or the current logical record.

Note

When you perform an Update Chunk operation following a Get operation, do *not* pass to the Update Chunk operation a Key Number that differs from the one specified in the preceding Get operation. If you do, the positioning established by the MicroKernel is unpredictable.

Version (26)

For client applications, the Version operation returns the local MicroKernel version and the Requester version, if applicable. If a client application opens a file on a server or specifies a server file path name in the Key Buffer, the Version operation also returns the MicroKernel version on that server. For server-based applications, the Version operation returns the server-based MicroKernel version and revision numbers.

Parameters

	Operation Code	Position Block	Data Buffer	Data Buffer Length	Key Buffer	Key Number
Sent	X			X		
Returned			X	X		

Prerequisites

Either the MicroKernel or the Requester must be loaded before you can issue a Version operation.

Procedure

1. Set the Operation Code to 26.

2. Set the Data Buffer Length to at least 15. (For more information, see Table 2-29.)

3. To retrieve the version number of a server-based MicroKernel, you must specify either a valid Position Block for an opened file on that server or a valid pathname in the Key Buffer.

Result

If you have both a workstation MicroKernel and client Requester configured for access and the Version operation is successful, the operation returns the version information for the workstation MicroKernel, the client Requester, and the server-based MicroKernel. Specify a 15-byte Data Buffer and Data Buffer Length. If both the client Requester and the workstation MicroKernel are loaded and you specify only a 5-byte Data Buffer and Data Buffer Length, the operation returns only the client Requester's version information.

In the Data Buffer, the Version operation returns a 5-byte Version Block for each MicroKernel or Requester, according to the format shown in . The fifth byte of each block identifies each MicroKernel or Requester.

Table 2-29 Version Block

Element	Length (in Bytes)	Description
Version Number	2	Btrieve version number.
Revision Number	2	Btrieve revision number.

Table 2-29   Version Block *continued*

Element	Length (in Bytes)	Description
Requester or Engine Type	1	Type of engine or requester; one of the following:
		◆ 9 (0x44) for Windows NT or Windows 95 workstation
		◆ 9 | 0x80 for Windows NT or Windows 95 workstation Developer Kit
		◆ D (0x39) for DOS workstation
		◆ L (0x4C) for Warp ServerO (0x4F) for OS/2 workstation
		◆ N (0x4E) for client Requester
		◆ S (0x53) for NetWare server
		◆ T (0x54) for Windows NT server
		◆ W (0x57) for Windows 3.x workstation
		◆ W | 0x80 for Windows 3.x workstation Developer Kit

For example, if you are running only Btrieve for NetWare 7.0, the Version operation returns the following hexadecimal values in the Data Buffer:

```
07 00 00 00 53
```

Byte-swapping as appropriate, the hexadecimal values become as follows:

```
00 07 00 00 53
```

After converting these values to decimal, the version number is 7, the revision number is 0, and the identifying component character is S (the letter equivalent of ASCII 0x53).

If the Version operation is unsuccessful, the MicroKernel returns a nonzero status code.

Positioning

The Version operation has no effect on positioning.

appendix A Language Interfaces

This appendix discusses the following language interfaces:

- ◆ "C/C++"
- ◆ "Cobol"
- ◆ "Delphi"
- ◆ "Pascal"
- ◆ "Visual Basic"

Table A-1 provides a list of the language interface source modules provided in the Programming Interfaces installation option. Pervasive Software provides the source code for each language interface. You can find additional information in the source modules.

If your programming language does not have an interface, check if your compiler supports mixed language calls. If so, you may be able to use the C interface.

Table A-1   Language Interface Source Modules

Language	Compiler	Source Module
C/C++	◆ Most C/C++ compilers, including Borland, Microsoft, and WATCOM. This interface provides multiple platform support.	◆ BlobHdr.h (for Extended DOS platforms using Borland or Phar Lap only)
		◆ BMemCopy.obj (for Extended DOS platforms using Borland or Phar Lap only)
		◆ BMemCopy.asm (source for BMemCopy.obj)
		◆ BtiTypes.h (platform-independent data types)
		◆ BtrApi.h (Btrieve function prototypes)
		◆ BtrApi.c (Btrieve interface code for all platforms)
		◆ BtrConst.h (common Btrieve constants)
		◆ BtrSamp.c (sample program)
		◆ GenStat.h (Pervasive status codes)
	◆ Borland C++ Builder	◆ CBBtrv.cpp
		◆ CBBtrv.mak
		◆ CBBMain.cpp
		◆ CBBMain.dfm
		◆ CBBMain.h

Table A-1   Language Interface Source Modules *continued*

Language	Compiler	Source Module
[Cobol]	◆ Micro Focus Cobol *all versions*	◆ MfxBtrv.bin (DOS Runtime for Cobol Animator, non-Intel byte-order integer) ◆ MfxBtrv.asm (source for this binary)
	◆ Microsoft Cobol 3	◆ CobrBtrv.obj (DOS 16-bit) ◆ CobpBtrv.obj (OS/2 16-bit) ◆ CobBtrv.asm (source for these objects) ◆ Mf2Btrv.bin (DOS runtime for Cobol animator, Intel byte-order integer) ◆ Mf2Btrv.asm (source for this binary) ◆ BtrSamp.cbl (sample program)

Table A-1   Language Interface Source Modules *continued*

Language	Compiler	Source Module
Delphi	• Borland Delphi 1	• Btr16.dpr
		• BtrSam16.pas (sample program)
		• BtrSam16.dfm
		• BtrApi16.pas
		• BtrConst.pas (common Btrieve constants)
	• Borland Delphi 3	• Btr32.dpr
		• Btr32.dof
		• BtrSam32.dfm
		• BtrSam32.pas (sample program)
		• BtrApi32.pas
		• BtrConst.pas (common Btrieve constants)
Pascal	• Borland Turbo Pascal 5–6	• BtrApid.pas
	• Borland Pascal 7 for DOS	• BtrSampd.pas (sample program)
	• Extended DOS Pascal for Turbo Pascal 7	• BtrConst.pas (common Btrieve constants)
		• BlobHdr.pas
	• Borland Turbo Pascal 1.5	• BtrApiw.pas
	• Borland Pascal 7 for Windows	• BtrSampw.pas (sample program)
		• BtrConst.pas (common Btrieve constants)
		• BlobHdr.pas

Table A-1   Language Interface Source Modules *continued*

Language	Compiler	Source Module
Visual Basic	◆ Microsoft Visual Basic for Windows 3.1	◆ BtSamp16.vbp ◆ BtrSam16.bas (sample program) ◆ BtrFrm16.frm
	◆ Microsoft Visual Basic for Windows NT and Windows 95	◆ BtSamp32.vbp ◆ BtrSam32.bas (sample program) ◆ BtrFrm32.frm

The following table provides a comparison of some common data types used in data buffers for Btrieve operations, such as Create and Stat.

Table A-2   Common Data Types Used in the Btrieve Data Buffer

Assembly	C	Cobol	Delphi	Pascal	Visual Basic
doubleword	long[1]	PIC 9(4)	longint*	longint*	Long integer
word	short int*	PIC 9(2)	smallint*	integer*	Integer
byte	char	PIC X	char	char	String
byte	unsigned char	PIC X	byte	byte	Byte

1 The value of integers depends on the environment in which you develop. In 32-bit environments, integers are the same as long integers. In 16-bit environments, integers are the same as short, or small, integers.

C/C++

The C/C++ interface facilitates writing platform-independent applications. This interface supports development for the following operating systems: DOS (including Extended DOS), NetWare Loadable Modules (NLMs), OS/2, Windows (including Win32 applications), Windows NT, and Windows 95.

This section discusses the following topics:

Interface Modules

This section describes in detail the modules that comprise the C language interface.

BtrApi.c

The file BtrApi.c is the actual implementation of the C application interface. It provides support for all applications that call BTRV and BTRVID, except NLMs, which do not require any interface code. When making a Btrieve call with either of these functions, compile BtrApi.c and link its object with the other modules in your application.

The BtrApi.c file contains #include directives that instruct your compiler to include BtrApi.h, BtrConst.h, BlobHdr.h, and BtiTypes.h. By including these files, BtrApi.c takes advantage of the data types that provide the platform independence associated with the interface.

BtrApi.h

The file BtrApi.h contains the prototypes of the Btrieve functions. The prototype definitions use the platform-independent data types defined in the file BtiTypes.h. BtrApi.h provides support for all applications calling the BTRV and BTRVID functions.

BtrConst.h

The file BtrConst.h contains useful constants specific to Btrieve. These constants can help you standardize references to Btrieve operation codes, status codes, file specification flags, key specification flags, and many more items.

You can use the C application interface without taking advantage of BtrConst.h; however, including the file may simplify your programming effort.

BtiTypes.h

The file BtiTypes.h defines the platform-independent data types. By using the data types in BtiTypes.h on your Btrieve function calls, your application ports among operating systems. See BtiTypes.h for more information about how the data types are defined for each operating system.

BtiTypes.h also describes the switches you must use to indicate the operating system on which your application runs. Table A-3 lists these operating system switches.

Table A-3   Operating System Switches

Operating System	Application Type	Switch
DOS	16-bit	BTI_DOS
	32-bit with Tenberry Extender and BStub.exe[1]	BTI_DOS_32R
	32-bit with Phar Lap 6	BTI_DOS_32P
	32-bit with Borland PowerPack	BTI_DOS_32B
NetWare	32-bit NLM	BTI_NLM
OS/2	16-bit	BTI_OS2
	32-bit	BTI_OS2_32
Win16	16-bit	BTI_WIN
Win32	32-bit	BTI_WIN_32

1 For more information, refer to "Creating Applications with the Tenberry DOS/4G Extender".

BtrSamp.c

The source file BtrSamp.c is a sample Btrieve program that you can compile, link, and run on any of the operating systems described in Table A-3.

Programming Requirements

If you use the C application interface to make your application platform independent, you must use the data types described in BtiTypes.h. To see how these data types are used, see the file BtrSamp.c.

Note

You must also specify a directive that identifies the operating system on which the program executes. The available values for the directive are listed in the header file BtiTypes.h. Specify the directive using the appropriate command line option for your compiler.

Program Example

The following example program, which is included on your distribution media, shows how to perform several of the more common Btrieve operations, and it performs those operations in the order required by the MicroKernel's dependencies (for example, you must open a file before performing I/O to it).

Note

Windows 3.x developers: The example uses the printf function to display text in a generic window. All of the supported C/C++ compilers are capable of building a generic window to display the output of Windows programs that employ standard I/O. By using standard I/O, the example allows you to concentrate on the details of programming a Btrieve application.

Note

16-bit DOS developers: To compile the example, you must have a stack size of at least 8K. To run the example, you must load the DOS Requester with the /T:1 flag.

Example A-1   BtrSamp.c

```
/*********************************************************
**
**    Copyright 1998 Pervasive Software Inc. All Rights Reserved
**
*********************************************************/
/*********************************************************
   BTRSAMP.C
 This is a simple sample designed to allow you to confirm your
 ability to compile, link, and execute a Btrieve application
for your target environment using your compiler tools.

 This program demonstrates the C/C++ interface for Btrieve for
DOS,Extended DOS, OS2 (16 and 32-bit), MS Windows, NetWare NLM,
MS Windows NT and Windows 95.

    This program does the following operations on the sample
database:
    - gets the Microkernel Database Engine version
    - opens sample.btr
    - gets a record on a known value of Key 0
    - displays the retrieved record
    - performs a stat operation
    - creates an empty 'clone' of sample.btr and opens it
    - performs a 'Get Next Extended' operation to extract a
subset of the records in sample.btr
    - inserts those records into the cloned file
    - closes both files
```

```
     IMPORTANT:
    You must specify the complete path to the directory that
contains the sample Btrieve data file, 'sample.btr'.  See
IMPORTANT, below.

   You can compile and run this program on any of the platforms
  supported by the interface modules.  Platforms are indicated
by the platform switches listed in 'btrapi.h'. For MS Windows
you should make an application that allows standard output via
printf().  Note that most C/C++ compilers support standard I/
O Windows applications.
    For MS Windows NT or OS2, you should make a console
application.

    See the prologue in 'btrapi.h' for information on how to
select a target platform for your application.  You must
specify a target platform.

***********************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <btrapi.h>
#include <btrconst.h>

/***********************************************************
  Constants
***********************************************************/
/***********************************************************
    IMPORTANT: You should modify the following to specify the
```

```
             complete path to 'sample.btr' for your environment.
***********************************************************/
#ifdef BTI_NLM
#define FILE1_NAME "sys:\\pvsw\\samples\\sample.btr"
#else
#define FILE1_NAME "c:\\pvsw\\samples\\sample.btr"
#endif

#ifdef BTI_NLM
#define FILE2_NAME "sys:\\pvsw\\samples\\sample2.btr"
#else
#define FILE2_NAME "c:\\pvsw\\samples\\sample2.btr"
#endif

#define EXIT_WITH_ERROR      1
#define TRUE                 1
#define FALSE                0
#define VERSION_OFFSET       0
#define REVISION_OFFSET      2
#define PLATFORM_ID_OFFSET   4
#define MY_THREAD_ID         50

/* Don't pad our structures */
#if defined(__BORLANDC__)
  #pragma option -a-
#else
  #if defined(_MSC_VER) || defined(__WATCOMC__)
    #pragma pack(1)
  #endif
#endif
```

```
/************************************************************
  Type definitions for Client ID and version Structures
************************************************************/
typedef struct
{
  BTI_CHAR networkAndNode[12];
  BTI_CHAR applicationID[2];
  BTI_WORD threadID;
} CLIENT_ID;

typedef struct
{
  BTI_SINT  Version;
  BTI_SINT  Revision;
  BTI_CHAR  MKDEId;
} VERSION_STRUCT;

/************************************************************
  Definition of record from 'sample.btr'
************************************************************/
typedef struct
{
  BTI_LONG  ID;
  BTI_CHAR  FirstName[16];
  BTI_CHAR  LastName[26];
  BTI_CHAR  Street[31];
  BTI_CHAR  City[31];
  BTI_CHAR  State[3];
  BTI_CHAR  Zip[11];
```

```
  BTI_CHAR  Country[21];
  BTI_CHAR  Phone[14];
} PERSON_STRUCT;

/**********************************************************
  Type definitions for Stat/Create structure
**********************************************************/
typedef struct
{
  BTI_SINT recLength;
  BTI_SINT pageSize;
  BTI_SINT indexCount;
  BTI_CHAR reserved[4];
  BTI_SINT flags;
  BTI_BYTE dupPointers;
  BTI_BYTE notUsed;
  BTI_SINT allocations;
} FILE_SPECS;

typedef struct
{
  BTI_SINT position;
  BTI_SINT length;
  BTI_SINT flags;
  BTI_CHAR reserved[4];
  BTI_CHAR type;
  BTI_CHAR null;
  BTI_CHAR notUsed[2];
  BTI_BYTE manualKeyNumber;
  BTI_BYTE acsNumber;
```

Example A-1   BtrSamp.c *continued*

```
} KEY_SPECS;

typedef struct
{
  FILE_SPECS fileSpecs;
  KEY_SPECS  keySpecs[5];
} FILE_CREATE_BUF;

/**********************************************************
   Structure type definitions for Get Next Extended operation
 **********************************************************/
typedef struct
{
  BTI_SINT    descriptionLen;
  BTI_CHAR    currencyConst[2];
  BTI_SINT    rejectCount;
  BTI_SINT    numberTerms;
} GNE_HEADER;

typedef struct
{
  BTI_CHAR    fieldType;
  BTI_SINT    fieldLen;
  BTI_SINT    fieldOffset;
  BTI_CHAR    comparisonCode;
  BTI_CHAR    connector;
  BTI_CHAR value[3];
} TERM_HEADER;

typedef struct
```

Example A-1 BtrSamp.c *continued*

```
{
  BTI_SINT    maxRecsToRetrieve;
  BTI_SINT    noFieldsToRetrieve;
} RETRIEVAL_HEADER;

typedef struct
{
  BTI_SINT    fieldLen;
  BTI_SINT    fieldOffset;
} FIELD_RETRIEVAL_HEADER;

typedef struct
{
  GNE_HEADER              gneHeader;
  TERM_HEADER             term1;
  TERM_HEADER             term2;
  RETRIEVAL_HEADER        retrieval;
  FIELD_RETRIEVAL_HEADER  recordRet;
} PRE_GNE_BUFFER;

typedef struct
{
  BTI_SINT     recLen;
  BTI_LONG     recPos;
  PERSON_STRUCT personRecord;
} RETURNED_REC;

typedef struct
{
  BTI_SINT        numReturned;
```

```
  RETURNED_REC  recs[20];
} POST_GNE_BUFFER;

typedef union
{
  PRE_GNE_BUFFER  preBuf;
  POST_GNE_BUFFER postBuf;
} GNE_BUFFER, BTI_FAR* GNE_BUFFER_PTR;

/* restore structure packing */
#if defined(__BORLANDC__)
  #pragma option -a.
#else
  #if defined(_MSC_VER) || defined(__WATCOMC__)
    #pragma pack()
  #endif
#endif

/***********************************************************
  Main
***********************************************************/
int main(void)
{
  /* Btrieve function parameters */
  BTI_BYTE posBlock1[128];
  BTI_BYTE posBlock2[128];
  BTI_BYTE dataBuf[255];
  BTI_WORD dataLen;
  BTI_BYTE keyBuf1[255];
  BTI_BYTE keyBuf2[255];
```

```
  BTI_WORD keyNum = 0;

  BTI_BYTE btrieveLoaded = FALSE;
  BTI_LONG personID;
  BTI_BYTE file1Open = FALSE;
  BTI_BYTE file2Open = FALSE;
  BTI_SINT status;
  BTI_SINT getStatus = -1;
  BTI_SINT i;
  BTI_SINT posCtr;

  CLIENT_ID       clientID;
  VERSION_STRUCT  versionBuffer[3];
  FILE_CREATE_BUF fileCreateBuf;
  GNE_BUFFER_PTR  gneBuffer;
  PERSON_STRUCT   personRecord;

  printf("**************** Btrieve C/C++ Interface Demo
***************\n\n");

  /* set up the Client ID */
  memset(clientID.networkAndNode, 0,
sizeof(clientID.networkAndNode));
  memcpy(clientID.applicationID, "MT", 2);  /* must be greater
than "AA" */
  clientID.threadID = MY_THREAD_ID;

  memset(versionBuffer, 0, sizeof(versionBuffer));
  dataLen = sizeof(versionBuffer);
```

```
status = BTRVID(
            B_VERSION,
            posBlock1,
            &versionBuffer,
            &dataLen,
            keyBuf1,
            keyNum,
            (BTI_BUFFER_PTR)&clientID);

if (status == B_NO_ERROR)
{
  printf("Btrieve Versions returned are: \n");
  for (i = 0; i < 3; i++) {
    if (versionBuffer[i].Version != 0)
    {
      printf("   %d.%d %c\n", versionBuffer[i].Version,
              versionBuffer[i].Revision,
              versionBuffer[i].MKDEId);
    }
  }
  printf("\n");
  btrieveLoaded = TRUE;
}
else
{
  printf("Btrieve B_VERSION status = %d\n", status);
  if (status != B_RECORD_MANAGER_INACTIVE)
  {
    btrieveLoaded = TRUE;
  }
```

```
   }

   /* clear buffers */
   if (status == B_NO_ERROR)
   {
     memset(dataBuf, 0, sizeof(dataBuf));
     memset(keyBuf1, 0, sizeof(keyBuf1));
     memset(keyBuf2, 0, sizeof(keyBuf2));
   }

   /* open sample.btr */
   if (status == B_NO_ERROR)
   {

     strcpy((BTI_CHAR *)keyBuf1, FILE1_NAME);
     strcpy((BTI_CHAR *)keyBuf2, FILE2_NAME);

     keyNum  = 0;
     dataLen = 0;

     status = BTRVID(
                 B_OPEN,
                 posBlock1,
                 dataBuf,
                 &dataLen,
                 keyBuf1,
                 keyNum,
                 (BTI_BUFFER_PTR)&clientID);
```

```
     printf("Btrieve B_OPEN status (sample.btr) = %d\n",
status);
    if (status == B_NO_ERROR)
    {
      file1Open = TRUE;
    }
  }

  /* get the record with key 0 = 263512477 using B_GET_EQUAL */
  if (status == B_NO_ERROR)
  {
    memset(&personRecord, 0, sizeof(personRecord));
    dataLen = sizeof(personRecord);
   personID = 263512477;   /* this is really a social security
number */
    *(BTI_LONG BTI_FAR *)&keyBuf1[0] = personID;
    keyNum = 0;

    status = BTRVID(
                B_GET_EQUAL,
                posBlock1,
                &personRecord,
                &dataLen,
                keyBuf1,
                keyNum,
                (BTI_BUFFER_PTR)&clientID);

    printf("Btrieve B_GET_EQUAL status = %d\n", status);
    if (status == B_NO_ERROR)
    {
```

```
      printf("\n");
      printf("The retrieved record is:\n");
      printf("ID:      %ld\n", personRecord.ID);
      printf("Name:    %s %s\n", personRecord.FirstName,
                                  personRecord.LastName);
      printf("Street:  %s\n", personRecord.Street);
      printf("City:    %s\n", personRecord.City);
      printf("State:   %s\n", personRecord.State);
      printf("Zip:     %s\n", personRecord.Zip);
      printf("Country: %s\n", personRecord.Country);
      printf("Phone:   %s\n", personRecord.Phone);
      printf("\n");
   }
}

/* perform a stat operation to populate the create buffer */
memset(&fileCreateBuf, 0, sizeof(fileCreateBuf));
dataLen = sizeof(fileCreateBuf);
keyNum = (BTI_WORD)-1;

status = BTRVID(B_STAT,
               posBlock1,
               &fileCreateBuf,
               &dataLen,
               keyBuf1,
               keyNum,
               (BTI_BUFFER_PTR)&clientID);

if (status == B_NO_ERROR)
{
```

```
    /* create and open sample2.btr */
    keyNum = 0;
    dataLen = sizeof(fileCreateBuf);
    status = BTRVID(B_CREATE,
                    posBlock2,
                    &fileCreateBuf,
                    &dataLen,
                    keyBuf2,
                    keyNum,
                    (BTI_BUFFER_PTR)&clientID);

    printf("Btrieve B_CREATE status = %d\n", status);
  }

  if (status == B_NO_ERROR)
  {
    keyNum = 0;
    dataLen = 0;

    status = BTRVID(
                    B_OPEN,
                    posBlock2,
                    dataBuf,
                    &dataLen,
                    keyBuf2,
                    keyNum,
                    (BTI_BUFFER_PTR)&clientID);

    printf("Btrieve B_OPEN status (sample2.btr) = %d\n",
 status);
```

```
    if (status == B_NO_ERROR)
    {
      file2Open = TRUE;
    }
  }

  /* now extract data from the original file, insert into new
one */
  if (status == B_NO_ERROR)
  {
    /* getFirst to establish currency */
    keyNum = 2; /* STATE-CITY index */
    memset(&personRecord, 0, sizeof(personRecord));
    memset(&keyBuf2[0], 0, sizeof(keyBuf2));
    dataLen = sizeof(personRecord);

    getStatus = BTRVID(
                  B_GET_FIRST,
                  posBlock1,
                  &personRecord,
                  &dataLen,
                  keyBuf1,
                  keyNum,
                  (BTI_BUFFER_PTR)&clientID);

    printf("Btrieve B_GET_FIRST status (sample.btr) = %d\n\n",
getStatus);
  }

  gneBuffer = malloc(sizeof(GNE_BUFFER));
```

```
   if (gneBuffer == NULL)
   {
      printf("Insufficient memory to allocate buffer");
     return(EXIT_WITH_ERROR);
   }
  memset(gneBuffer, 0, sizeof(GNE_BUFFER));
 memcpy(&gneBuffer->preBuf.gneHeader.currencyConst[0], "UC",
2);
  while (getStatus == B_NO_ERROR)
  {
    gneBuffer->preBuf.gneHeader.rejectCount = 0;
    gneBuffer->preBuf.gneHeader.numberTerms = 2;
    posCtr = sizeof(GNE_HEADER);

    /* fill in the first condition */
    gneBuffer->preBuf.term1.fieldType = 11;
    gneBuffer->preBuf.term1.fieldLen = 3;
    gneBuffer->preBuf.term1.fieldOffset = 108;
    gneBuffer->preBuf.term1.comparisonCode = 1;
    gneBuffer->preBuf.term1.connector = 2;
    memcpy(&gneBuffer->preBuf.term1.value[0], "TX", 2);
    posCtr += sizeof(TERM_HEADER);

    /* fill in the second condition */
    gneBuffer->preBuf.term2.fieldType = 11;
    gneBuffer->preBuf.term2.fieldLen = 3;
    gneBuffer->preBuf.term2.fieldOffset = 108;
    gneBuffer->preBuf.term2.comparisonCode = 1;
    gneBuffer->preBuf.term2.connector = 0;
    memcpy(&gneBuffer->preBuf.term2.value[0], "CA", 2);
```

```
     posCtr += sizeof(TERM_HEADER);

    /* fill in the projection header to retrieve whole record */
    gneBuffer->preBuf.retrieval.maxRecsToRetrieve = 20;
    gneBuffer->preBuf.retrieval.noFieldsToRetrieve = 1;
    posCtr += sizeof(RETRIEVAL_HEADER);
    gneBuffer->preBuf.recordRet.fieldLen =
sizeof(PERSON_STRUCT);
    gneBuffer->preBuf.recordRet.fieldOffset = 0;
    posCtr += sizeof(FIELD_RETRIEVAL_HEADER);
    gneBuffer->preBuf.gneHeader.descriptionLen = posCtr;

    dataLen = sizeof(GNE_BUFFER);
    getStatus = BTRVID(
                    B_GET_NEXT_EXTENDED,
                    posBlock1,
                    gneBuffer,
                    &dataLen,
                    keyBuf1,
                    keyNum,
                    (BTI_BUFFER_PTR)&clientID);

     printf("Btrieve B_GET_NEXT_EXTENDED status = %d\n",
getStatus);

    /* Get Next Extended can reach end of file and still return
some records */
    if ((getStatus == B_NO_ERROR) || (getStatus ==
B_END_OF_FILE))
     {
```

```
       printf("GetNextExtended returned %d records.\n",
gneBuffer->postBuf.numReturned);
       for (i = 0; i < gneBuffer->postBuf.numReturned; i++)
       {
         dataLen = sizeof(PERSON_STRUCT);
         memcpy(dataBuf, &gneBuffer-
>postBuf.recs[i].personRecord, dataLen);
         status = BTRVID(
                    B_INSERT,
                    posBlock2,
                    dataBuf,
                    &dataLen,
                    keyBuf2,
                    -1,   /* no currency change */
                    (BTI_BUFFER_PTR)&clientID);
       }
     printf("Inserted %d records in new file, status = %d\n\n",
gneBuffer->postBuf.numReturned, status);
     }
    memset(gneBuffer, 0, sizeof(GNE_BUFFER));
    memcpy(&gneBuffer->preBuf.gneHeader.currencyConst[0],
"EG", 2);
  }

  free(gneBuffer);
  gneBuffer = NULL;

  /* close open files */
  if (file1Open)
  {
```

```
    dataLen = 0;

    status = BTRVID(
                B_CLOSE,
                posBlock1,
                dataBuf,
                &dataLen,
                keyBuf1,
                keyNum,
                (BTI_BUFFER_PTR)&clientID);

    printf("Btrieve B_CLOSE status (sample.btr) = %d\n",
status);
  }

  if (file2Open)
  {
    dataLen = 0;

    status = BTRVID(
                B_CLOSE,
                posBlock2,
                dataBuf,
                &dataLen,
                keyBuf2,
                keyNum,
                (BTI_BUFFER_PTR)&clientID);

    printf("Btrieve B_CLOSE status (sample2.btr) = %d\n",
status);
```

Example A-1   BtrSamp.c *continued*

```
   }

   /**********************************************************
      ISSUE THE BTRIEVE STOP OPERATION - For multi-tasking
environments, such as MS Windows, OS2, and NLM, 'stop' frees
all Btrieve resources for this client.  In DOS and Extended
DOS, it removes the Btrieve engine from memory, which we choose
not to do in this example.  In multi-tasking environments, the
engine will not unload on 'stop' unless it has no more clients.
   **********************************************************/
   #if !defined(BTI_DOS) && !defined(BTI_DOS_32R) && \
   !defined(BTI_DOS_32B) && !defined(BTI_DOS_32P)
   if (btrieveLoaded)
   {
     dataLen = 0;
     status = BTRVID(B_STOP, posBlock1, dataBuf, &dataLen,
keyBuf1, keyNum,
               (BTI_BUFFER_PTR)&clientID);
     printf("Btrieve STOP status = %d\n", status);
     if (status != B_NO_ERROR)
     {
        status = EXIT_WITH_ERROR;
     }
   }
   #endif

   return(status);
}
```

Creating Applications with the Tenberry DOS/4G Extender

Using the Tenberry extended DOS switch for your 32-bit extended DOS application provides optimum performance, because doing so frees your application from having to copy data to real-mode memory in order to pass the data to the MicroKernel.

To use the extended DOS switch, follow these steps:

1. Compile your 32-bit application and include the C interface. Be sure to define the BTI_DOS_32R macro to your compiler.

2. Run a compiler utility on the object modules to make them compatible with the Tenberry linker. (For the WATCOM compiler, this utility is called womp. The 10.0 compiler does not require womp.)

3. Create a .DEF file, as in the following example used to create the sample program BtrSamp.exe:

```
LIBRARY btrsamp.dll

DATA NONSHARED

EXPORTS
 __ImportedFunctions_
```

4. Link your application using the Tenberry linker (Glu) and specify the Btrieve stub, as in the following example:

```
glu -format lin -stack 40000 -stub bstub.exe -cod
```

Compiling, Linking, and Running the Program Example

The C Interface for Btrieve is platform-independent and compiler-independent. In general for your compiler, you must do the following:

◆ In the BtrSamp.c file, edit the paths to the Sample.btr and Sample2.btr files as appropriate.

◆ Identify the appropriate import library. The locations are as follows:

Microsoft   \Intf\ImpLib\W32

Watcom and Borland   \Intf\ImpLib\W32x

◆ Set the appropriate platform switch. The switches are shown in <u>Table A-3</u>.

◆ Update the include directories setting. You must include the current directory.

◆ Use the appropriate structure alignment. The sample program contains a pragma for structure alignment. Because you must either use a similar pragma in your own application or set a compiler switch for structure alignment, the instructions in this section identify the appropriate compiler switch to use.

This section provides instructions for some common development environments. If your development environment is not discussed, you can adapt these instructions.

➤ **In Microsoft Visual C++, to compile, link, and run the program example:**

1. In the Developer Studio programming environment, choose **New** from the **File** menu.

2. In the **New** dialog, on the **Projects** tab, do the following:
   a. Set the type to Console Application. (BtrSamp runs from a DOS prompt.)
   b. Set the Name to BtrSamp.

    c. Set the location to the \Intf\C directory.

    d. Click OK.

    Developer Studio creates a new project called BtrSamp.

3. In the **FileView** tab of the workspace, right click on the BtrSamp project, choose the **Add Files to Project** command, and add the following files:

    • \Intf\C\BtrApi.c

    • \Intf\C\BtrSamp.c

    • \Intf\ImpLib\Win32\W3Btrv7.lib

    • Click OK.

4. In the BtrSamp.c file, edit the paths to the Sample.btr and Sample2.btr files as appropriate.

5. Choose **Settings** from the **Project** menu. On the **C++** tab, do the following:

    • In the Preprocessor category, add BTI_WIN_32 in the Preprocessor Definitions box and add the current directory (.) in the Additional Include Directories box.

    • In the Code Generation category, set the Structure Member Alignment to 1 Byte.

    • Click OK.

6. Choose **Rebuild All** from the **Build** menu.

    Microsoft Visual C++ builds the program example and places BtrSamp.exe in your project file directory.

7. In a DOS prompt window, run BtrSamp.exe.

➤ **In Watcom C++, to compile, link, and run the program example:**

1. In the IDE programming environment, choose **New Project** from the **File** menu.

2. In the **Enter Project Filename** dialog, set the File Name to BtrSamp.wpj and click OK.

3. In the **New Target** dialog, do the following:

- Set the Target Name to BtrSamp.

- Set the Target Environment to Win32.

- Set the Image Type to Executable [.exe].

- Click OK.

4. Choose **New Source** from the **Sources** menu and add the following files:

- \Intf\C\BtrApi.c

- \Intf\C\BtrSamp.c

- \Intf\ImpLib\Win32x\W3Btrv7.lib

- Click OK.

5. In the BtrSamp.c file, edit the paths to the Sample.btr and Sample2.btr files as appropriate.

6. Choose **C Compiler Switches** from the **Source Options** submenu on the **Sources** menu and do the following:

- In the **File Option Switches** dialog, add the current directory to the Include directories box, as in the following example:

  ```
  $(%watcom)\h;$(%watcom)\h\nt;.
  ```

- In the **Source Switches** dialog, add BTI_WIN_32 to the Macro Definitions box and set the Structure Alignment to Pack structures.

- Click OK.

7. Choose the **Make All Targets** button from the toolbar.

WATCOM C++ builds the program example and places BtrSamp.exe in your project file directory.

8. In a DOS prompt window, run BtrSamp.exe.

➤ **In Borland C++, to compile, link, and run the program example:**

1. In the Borland C++ programming environment, choose **New Project** from the **Project** menu.

2. In the **New Target** dialog:

- ◆ Set the Project Path and Name to \Intf\C\BtrSamp.

- ◆ Set the Target Name to BtrSamp.

- ◆ Set the Target Type to Application [.exe].

- ◆ Set the Target Model to Console.

- ◆ Turn off the Class Library checkbox in the Frameworks group.

- ◆ Click the Advanced button and specify .C node in the Initial Nodes group and turn off the .rc and .def check boxes.

- ◆ Click OK.

3. In the **Project** window, right click on the BtrSamp project, choose **Add Node**, and add the following files:

- ◆ \Intf\C\BtrApi.c

- ◆ \Intf\C\BtrSamp.c

- ◆ \Intf\ImpLib\Win32x\W3Btrv7.lib

- ◆ Click OK.

4. In the BtrSamp.c file, edit the paths to the Sample.btr and Sample2.btr files as appropriate.

5. Choose **Project** from the **Options** menu and do the following:

- Select the Directories topic and add the current directory to the Include list in the Source Directories group, as in the following example:

```
c:\bc5\include;.
```

- Select the Defines topic under Compiler and add BTI_WIN_32 to the Defines box.

- Select the Processor topic under 32-bit Compiler and set the Data Alignment to Byte.

- Click OK.

6. Choose **Build All** from the **Project** menu.

Borland C++ builds the program example and places BtrSamp.exe in your project file directory.

7. In a DOS prompt window, run BtrSamp.exe.

Compiling C++ Builder Applications

➤ **In C++ Builder, to compile, link, and run the program example:**

1. Choose **Open Project** from the **File** menu and select the CBBtrv.mak file.

2. Choose **Project** from the **Options** menu, select the Directories/Conditionals tab, and do the following:

   - Add the \Intf\C directory to the Include Path.

   - Add the \Intf\Implib\Win32x directory to the Library Path.

3. Choose **Make** from the **Project** menu.

   C++ Builder builds the program example and places CBBtrv.exe in your project file directory.

4. Choose **Run** from the **Run** menu.

5. In the Btrieve Sample Application window, click the Run Test button.

The following example C++ Builder program shows how to perform several of the more common Btrieve operations, and it performs those operations in the order required by the MicroKernel's dependencies (for example, you must open a file before performing I/O to it).

Example A-2   CBBMain.cpp

```
/***********************************************************
**
** Copyright 1998 Pervasive Software Inc.
** All Rights Reserved
***********************************************************/
/***********************************************************
  CBBMain.cpp
 This is a simple sample designed to allow you to confirm your
 ability to compile, link, and execute a Btrieve application
```

Example A-2   CBBMain.cpp

```
     for your target environment using your compiler tools.

 This program demonstrates the C/C++ interface for Btrieve for
 MS Windows NT and Windows 95 with Borland's C++ Builder.

 This program does the following operations on the sample
database:
 - gets the Microkernel Database Engine version
 - opens sample.btr
 - gets a record on a known value of Key 0
 - displays the retrieved record
 - performs a stat operation
 - creates an empty 'clone' of sample.btr and opens it
 - performs a 'Get Next Extended' operation to extract a subset
         of the records in sample.btr
 - inserts those records into the cloned file
 - closes both files

 To compile this sample, you must make the following files from
 the Btrieve "C" interface directory available to the IDE:
    - btrapi.c
    - btrapi.h
    - btrconst.h

IMPORTANT:
You must specify the complete path to the directory that
contains the sample Btrieve data file, 'sample.btr'.  See
IMPORTANT, below.

*********************************************************/
//-----------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "CBBMain.h"
#include <btrapi.h>
#include <btrconst.h>
```

Example A-2   CBBMain.cpp

```
//-----------------------------------------------------------
#pragma resource "*.dfm"

/***********************************************************
   Constants
 ***********************************************************/
#define EXIT_WITH_ERROR      1
#define TRUE                 1
#define FALSE                0
#define VERSION_OFFSET       0
#define REVISION_OFFSET      2
#define PLATFORM_ID_OFFSET   4
#define MY_THREAD_ID         50


/* Don't pad our structures */
/* Borland's help says this is the default. Don't believe it.*/
#pragma option -a1

/***********************************************************
   Type definitions for Client ID and version Structures
 ***********************************************************/
typedef struct
{
  BTI_CHAR networkAndNode[12];
  BTI_CHAR applicationID[2];
  BTI_WORD threadID;
} CLIENT_ID;

typedef struct
{
  BTI_SINT  Version;
  BTI_SINT  Revision;
  BTI_CHAR  MKDEId;
} VERSION_STRUCT;

/***********************************************************
```

```
   Definition of record from 'sample.btr'
 ***********************************************************/
typedef struct
{
  BTI_LONG  ID;
  BTI_CHAR  FirstName[16];
  BTI_CHAR  LastName[26];
  BTI_CHAR  Street[31];
  BTI_CHAR  City[31];
  BTI_CHAR  State[3];
  BTI_CHAR  Zip[11];
  BTI_CHAR  Country[21];
  BTI_CHAR  Phone[14];
} PERSON_STRUCT;

/***********************************************************
   Type definitions for Stat/Create structure
 ***********************************************************/
typedef struct
{
  BTI_SINT recLength;
  BTI_SINT pageSize;
  BTI_SINT indexCount;
  BTI_CHAR reserved[4];
  BTI_SINT flags;
  BTI_BYTE dupPointers;
  BTI_BYTE notUsed;
  BTI_SINT allocations;
} FILE_SPECS;

typedef struct
{
  BTI_SINT position;
  BTI_SINT length;
  BTI_SINT flags;
  BTI_CHAR reserved[4];
  BTI_CHAR type;
```

Example A-2   CBBMain.cpp

```
  BTI_CHAR null;
  BTI_CHAR notUsed[2];
  BTI_BYTE manualKeyNumber;
  BTI_BYTE acsNumber;
} KEY_SPECS;

typedef struct
{
  FILE_SPECS fileSpecs;
  KEY_SPECS  keySpecs[5];
} FILE_CREATE_BUF;

/***********************************************************
  Structure type definitions for Get Next Extended operation
***********************************************************/
typedef struct
{
  BTI_SINT    descriptionLen;
  BTI_CHAR    currencyConst[2];
  BTI_SINT    rejectCount;
  BTI_SINT    numberTerms;
} GNE_HEADER;

typedef struct
{
  BTI_CHAR    fieldType;
  BTI_SINT    fieldLen;
  BTI_SINT    fieldOffset;
  BTI_CHAR    comparisonCode;
  BTI_CHAR    connector;
  BTI_CHAR value[3];
} TERM_HEADER;

typedef struct
{
  BTI_SINT    maxRecsToRetrieve;
  BTI_SINT    noFieldsToRetrieve;
```

Example A-2   CBBMain.cpp

```
} RETRIEVAL_HEADER;

typedef struct
{
  BTI_SINT    fieldLen;
  BTI_SINT    fieldOffset;
} FIELD_RETRIEVAL_HEADER;

typedef struct
{
  GNE_HEADER               gneHeader;
  TERM_HEADER              term1;
  TERM_HEADER              term2;
  RETRIEVAL_HEADER         retrieval;
  FIELD_RETRIEVAL_HEADER   recordRet;
} PRE_GNE_BUFFER;

typedef struct
{
  BTI_SINT      recLen;
  BTI_LONG      recPos;
  PERSON_STRUCT personRecord;
} RETURNED_REC;

typedef struct
{
  BTI_SINT      numReturned;
  RETURNED_REC  recs[20];
} POST_GNE_BUFFER;

typedef union
{
  PRE_GNE_BUFFER  preBuf;
  POST_GNE_BUFFER postBuf;
} GNE_BUFFER, BTI_FAR* GNE_BUFFER_PTR;

//------------------------------------------------------------
```

Example A-2   CBBMain.cpp

```cpp
// non-exported forward declarations
void printLB(TListBox *LB, char *msg);

TForm1 *Form1;
//-----------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//-----------------------------------------------------------
void __fastcall TForm1::ExitButtonClick(TObject *Sender)
{
    Form1->Close();
}
//-----------------------------------------------------------
void __fastcall TForm1::RunButtonClick(TObject *Sender)
{
    runTest();
}

/***********************************************************
  Helper function to write to ListBox
***********************************************************/
void printLB(TListBox *LB, char *msg)
{
    LB->Items->Add(msg);
}


/***********************************************************
  This is where all the work gets done
***********************************************************/
BTI_SINT runTest(void)
{
  /* Btrieve function parameters */
  BTI_BYTE posBlock1[128];
  BTI_BYTE posBlock2[128];
```

Example A-2   CBBMain.cpp

```
   BTI_BYTE dataBuf[255];
   BTI_WORD dataLen;
   BTI_BYTE keyBuf1[255];
   BTI_BYTE keyBuf2[255];
   BTI_WORD keyNum = 0;

   BTI_BYTE btrieveLoaded = FALSE;
   BTI_LONG personID;
   BTI_BYTE file1Open = FALSE;
   BTI_BYTE file2Open = FALSE;
   BTI_SINT status;
   BTI_SINT getStatus;
   BTI_SINT i;
   BTI_SINT posCtr;

   CLIENT_ID       clientID;
   VERSION_STRUCT  versionBuffer[3];
   FILE_CREATE_BUF fileCreateBuf;
   GNE_BUFFER_PTR  gneBuffer;
   PERSON_STRUCT   personRecord;

   BTI_CHAR tmpBuf[1024];

   /**********************************************************
      Print the program title
   **********************************************************/
   printLB(Form1->ListBox1,
        "**** Btrieve -- C++ Builder Sample ****" );

   /* set up the Client ID */
   memset(clientID.networkAndNode, 0,
          sizeof(clientID.networkAndNode));
   memcpy(clientID.applicationID, "MT", 2);
 /* must be greater than "AA" */
   clientID.threadID = MY_THREAD_ID;
```

Example A-2   CBBMain.cpp

```cpp
    memset(versionBuffer, 0, sizeof(versionBuffer));
    dataLen = sizeof(versionBuffer);

    status = BTRVID(
                B_VERSION,
                posBlock1,
                &versionBuffer,
                &dataLen,
                keyBuf1,
                keyNum,
                (BTI_BUFFER_PTR)&clientID);

    if (status == B_NO_ERROR)
    {
      strcpy(tmpBuf, "Btrieve Versions returned are: " );
      printLB(Form1->ListBox1, tmpBuf);
      for (i = 0; i < 3; i++) {
        if (versionBuffer[i].Version != 0)
        {
        sprintf(tmpBuf, "   %d.%d %c", versionBuffer[i].Version,
                  versionBuffer[i].Revision,
                  versionBuffer[i].MKDEId );
        printLB(Form1->ListBox1, tmpBuf);
        }
      }
      printLB(Form1->ListBox1, "");
      btrieveLoaded = TRUE;
    }
    else
    {
      sprintf(tmpBuf, "Btrieve B_VERSION status = %d", status );
      printLB(Form1->ListBox1, tmpBuf);
      if ( status != B_RECORD_MANAGER_INACTIVE )
      {
        btrieveLoaded = TRUE;
      }
    }
```

Example A-2   CBBMain.cpp

```cpp
  /* clear buffers */
  if (status == B_NO_ERROR)
  {
    memset(dataBuf, 0, sizeof(dataBuf));
    memset(keyBuf1, 0, sizeof(keyBuf1));
    memset(keyBuf2, 0, sizeof(keyBuf2));
  }

  /* open sample.btr */
  if (status == B_NO_ERROR)
  {
    /*********************************************************
IMPORTANT: You should modify the following to specify the
complete path to 'sample.btr' for your environment.
    *********************************************************/
    strcpy((BTI_CHAR *)keyBuf1, "c:\\sample\\sample.btr");
    strcpy((BTI_CHAR *)keyBuf2, "c:\\sample\\sample2.btr");

    keyNum  = 0;
    dataLen = 0;

    status = BTRVID(
                B_OPEN,
                posBlock1,
                dataBuf,
                &dataLen,
                keyBuf1,
                keyNum,
                (BTI_BUFFER_PTR)&clientID);

    sprintf(tmpBuf, "Btrieve B_OPEN status = %d", status );
    printLB(Form1->ListBox1, tmpBuf);
    if (status == B_NO_ERROR)
    {
      file1Open = TRUE;
    }
```

Example A-2   CBBMain.cpp

```
    }

  /* get the record with key 0 = 263512477 using B_GET_EQUAL */
  if (status == B_NO_ERROR)
  {
    memset(&personRecord, 0, sizeof(personRecord));
    dataLen = sizeof(personRecord);
   personID = 263512477;   /* this is really a social security
number */
    *(BTI_LONG BTI_FAR *)&keyBuf1[0] = personID;
    keyNum = 0;

    status = BTRVID(
                 B_GET_EQUAL,
                 posBlock1,
                 &personRecord,
                 &dataLen,
                 keyBuf1,
                 keyNum,
                 (BTI_BUFFER_PTR)&clientID);

    sprintf(tmpBuf, "Btrieve B_GET_EQUAL status = %d",
               status );
    printLB(Form1->ListBox1, tmpBuf);
    if (status == B_NO_ERROR)
    {
      sprintf(tmpBuf, "" );
      printLB(Form1->ListBox1, tmpBuf);
      sprintf(tmpBuf, "The retrieved record is:" );
      printLB(Form1->ListBox1, tmpBuf);
      sprintf(tmpBuf, "ID:      %ld", personRecord.ID );
         printLB(Form1->ListBox1, tmpBuf);
     sprintf(tmpBuf, "Name:    %s %s", personRecord.FirstName,
                                      personRecord.LastName );
      printLB(Form1->ListBox1, tmpBuf);
      sprintf(tmpBuf, "Street:  %s", personRecord.Street );
```

Example A-2 CBBMain.cpp

```
      printLB(Form1->ListBox1, tmpBuf);
      sprintf(tmpBuf, "City:    %s", personRecord.City );
      printLB(Form1->ListBox1, tmpBuf);
      sprintf(tmpBuf, "State:   %s", personRecord.State );
      printLB(Form1->ListBox1, tmpBuf);
      sprintf(tmpBuf, "Zip:     %s", personRecord.Zip );
      printLB(Form1->ListBox1, tmpBuf);
      sprintf(tmpBuf, "Country: %s", personRecord.Country );
      printLB(Form1->ListBox1, tmpBuf);
      sprintf(tmpBuf, "Phone:   %s", personRecord.Phone );
      printLB(Form1->ListBox1, tmpBuf);
      sprintf(tmpBuf, "" );
      printLB(Form1->ListBox1, tmpBuf);
    }
  }

  /* perform a stat operation to populate the create buffer */

  memset(&fileCreateBuf, 0, sizeof(fileCreateBuf));
  dataLen = sizeof(fileCreateBuf);
  keyNum = (BTI_WORD)-1;

  status = BTRVID(B_STAT,
                  posBlock1,
                  &fileCreateBuf,
                  &dataLen,
                  keyBuf1,
                  keyNum,
                  (BTI_BUFFER_PTR)&clientID);

  if (status == B_NO_ERROR)
  {
    /* create and open sample2.btr */
    keyNum = 0;
    dataLen = sizeof(fileCreateBuf);
    status = BTRVID(B_CREATE,
                    posBlock2,
```

Example A-2   CBBMain.cpp

```
                       &fileCreateBuf,
                       &dataLen,
                       keyBuf2,
                       keyNum,
                       (BTI_BUFFER_PTR)&clientID);

     sprintf(tmpBuf, "Btrieve B_CREATE status = %d", status );
     printLB(Form1->ListBox1, tmpBuf);
   }

   if (status == B_NO_ERROR)
   {
     keyNum = 0;
     dataLen = 0;

     status = BTRVID(
                   B_OPEN,
                   posBlock2,
                   dataBuf,
                   &dataLen,
                   keyBuf2,
                   keyNum,
                   (BTI_BUFFER_PTR)&clientID);

     sprintf(tmpBuf, "Btrieve B_OPEN status = %d", status );
     printLB(Form1->ListBox1, tmpBuf);
     if (status == B_NO_ERROR)
     {
       file2Open = TRUE;
     }
   }
 /* now extract data from original file, insert into new one */
   if (status == B_NO_ERROR)
   {
     /* getFirst to establish currency */
     keyNum = 2; /* STATE-CITY index */
```

Example A-2   CBBMain.cpp

```cpp
    memset(&personRecord, 0, sizeof(personRecord));
    memset(&keyBuf2[0], 0, sizeof(keyBuf2));
    dataLen = sizeof(personRecord);

    getStatus = BTRVID(
                    B_GET_FIRST,
                    posBlock1,
                    &personRecord,
                    &dataLen,
                    keyBuf1,
                    keyNum,
                    (BTI_BUFFER_PTR)&clientID);

    sprintf(tmpBuf, "Btrieve B_GET_FIRST status = %d",
getStatus );
    printLB(Form1->ListBox1, tmpBuf);
  }

  gneBuffer = (GNE_BUFFER_PTR)malloc(sizeof(GNE_BUFFER));
  if (gneBuffer == NULL)
  {
   strcpy(tmpBuf, "Insufficient memory to allocate buffer" );
   printLB(Form1->ListBox1, tmpBuf);
   return(EXIT_WITH_ERROR);
  }
  memset(gneBuffer, 0, sizeof(GNE_BUFFER));
  memcpy(&gneBuffer->preBuf.gneHeader.currencyConst[0], "UC",
         2);
  while (getStatus == B_NO_ERROR)
  {
    gneBuffer->preBuf.gneHeader.rejectCount = 0;
    gneBuffer->preBuf.gneHeader.numberTerms = 2;
    posCtr = sizeof(GNE_HEADER);

    /* fill in the first condition */
    gneBuffer->preBuf.term1.fieldType = 11;
```

Example A-2   CBBMain.cpp

```
    gneBuffer->preBuf.term1.fieldLen = 3;
    gneBuffer->preBuf.term1.fieldOffset = 108;
    gneBuffer->preBuf.term1.comparisonCode = 1;
    gneBuffer->preBuf.term1.connector = 2;
    memcpy(&gneBuffer->preBuf.term1.value[0], "TX", 2);
    posCtr += sizeof(TERM_HEADER);

    /* fill in the second condition */
    gneBuffer->preBuf.term2.fieldType = 11;
    gneBuffer->preBuf.term2.fieldLen = 3;
    gneBuffer->preBuf.term2.fieldOffset = 108;
    gneBuffer->preBuf.term2.comparisonCode = 1;
    gneBuffer->preBuf.term2.connector = 0;
    memcpy(&gneBuffer->preBuf.term2.value[0], "CA", 2);
    posCtr += sizeof(TERM_HEADER);

   /* fill in the projection header to retrieve whole record */
    gneBuffer->preBuf.retrieval.maxRecsToRetrieve = 20;
    gneBuffer->preBuf.retrieval.noFieldsToRetrieve = 1;
    posCtr += sizeof(RETRIEVAL_HEADER);
    gneBuffer->preBuf.recordRet.fieldLen =
         sizeof(PERSON_STRUCT);
    gneBuffer->preBuf.recordRet.fieldOffset = 0;
    posCtr += sizeof(FIELD_RETRIEVAL_HEADER);
    gneBuffer->preBuf.gneHeader.descriptionLen = posCtr;

    dataLen = sizeof(GNE_BUFFER);
    getStatus = BTRVID(
                    B_GET_NEXT_EXTENDED,
                    posBlock1,
                    gneBuffer,
                    &dataLen,
                    keyBuf1,
                    keyNum,
                    (BTI_BUFFER_PTR)&clientID);
```

Example A-2   CBBMain.cpp

```
     sprintf(tmpBuf, "Btrieve B_GET_NEXT_EXTENDED status = %d",
           getStatus );
     printLB(Form1->ListBox1, tmpBuf);

/* Get Next Extended can reach end of file and still return */
/* some records */
     if ((getStatus == B_NO_ERROR) || (getStatus ==
               B_END_OF_FILE))
     {
       sprintf(tmpBuf, "GetNextExtended returned %d records.",
               gneBuffer->postBuf.numReturned);
      printLB(Form1->ListBox1, tmpBuf);
       for (i = 0; i < gneBuffer->postBuf.numReturned; i++)
       {
         dataLen = sizeof(PERSON_STRUCT);
         memcpy(dataBuf, &gneBuffer->
               postBuf.recs[i].personRecord, dataLen);
         status = BTRVID(
                     B_INSERT,
                     posBlock2,
                     dataBuf,
                     &dataLen,
                     keyBuf2,
                     -1,    /* no currency change */
                     (BTI_BUFFER_PTR)&clientID);
       }
       sprintf(tmpBuf, "Inserted %d records in new file,
               status = %d",
               gneBuffer->postBuf.numReturned, status);
      printLB(Form1->ListBox1, tmpBuf);
     }
     memset(gneBuffer, 0, sizeof(GNE_BUFFER));
     memcpy(&gneBuffer->preBuf.gneHeader.currencyConst[0],
         "EG", 2);
   }
```

Example A-2   CBBMain.cpp

```
   free(gneBuffer);
   gneBuffer = NULL;

   /* close open files */
   if (file1Open)
   {
     dataLen = 0;

     status = BTRVID(
                  B_CLOSE,
                  posBlock1,
                  dataBuf,
                  &dataLen,
                  keyBuf1,
                  keyNum,
                  (BTI_BUFFER_PTR)&clientID);

     sprintf(tmpBuf, "Btrieve B_CLOSE status (sample.btr) = %d",
          status );
     printLB(Form1->ListBox1, tmpBuf);
   }

   if (file2Open)
   {
     dataLen = 0;

     status = BTRVID(
                  B_CLOSE,
                  posBlock2,
                  dataBuf,
                  &dataLen,
                  keyBuf2,
                  keyNum,
                  (BTI_BUFFER_PTR)&clientID);
```

Example A-2   CBBMain.cpp

```
     sprintf(tmpBuf, "Btrieve B_CLOSE status (sample2.btr) =
          %d", status );
     printLB(Form1->ListBox1, tmpBuf);
   }

   /*********************************************************
  ISSUE THE BTRIEVE STOP OPERATION - For multi-tasking
  environments, such as MS Windows, OS2, and NLM, 'stop' frees
  all Btrieve resources for this client.  In DOS and Extended
  DOS, it removes the Btrieve engine from memory.  In multi-
  tasking environments, the engine will not unload on 'stop'
  unless it has no more clients.
   *********************************************************/
   if (btrieveLoaded)
   {
     dataLen = 0;
     status = BTRVID(B_STOP, posBlock1, dataBuf, &dataLen,
          keyBuf1, keyNum,(BTI_BUFFER_PTR)&clientID);
     sprintf(tmpBuf, "Btrieve B_STOP status = %d", status );
     printLB(Form1->ListBox1, tmpBuf);
     if (status != B_NO_ERROR)
     {
        status = EXIT_WITH_ERROR;
     }
   }

   return(status);
}
```

Cobol

Animated Cobol developers: The Cobol animator passes stack parameters from left to right, while the non-animated Cobol interface passes parameters from right to left. However, Cobol passes integers in the Intel high-low format for both animated and non-animated applications.

Also, the object modules MF2BtrV.OBJ & CSUPPORT.OBJ have been dropped from the Cobol interface. Use the module COBRBtrV.OBJ in place of these.

Non-animated Cobol developers: All numerical values passed to the MicroKernel as a parameter must be in the Intel format (low-high byte order). To accomplish this, define the values as COMP-5.

OS/2 non-animated developers: For an OS/2 module, define OS2=1 to your assembler. If you do not, then the assembled module is for DOS.

Program Example

The following example program, which is included on your distribution media, shows how to perform several of the more common Btrieve operations, and it performs those operations in the order required by the MicroKernel's dependencies (for example, you must open a file before performing I/O to it).

Example A-3   Btrsamp.cbl

```
*
*
*    Copyright 1998 Pervasive Software Inc. All Rights Reserved
*
```

Example A-3   Btrsamp.cbl *continued*

```
*
*  BTRSAMP.CBL
* This is a sample COBOL program that makes Btrieve calls
* from an application using Micro Focus COBOL v3.x.
*
*
       IDENTIFICATION DIVISION.
      *
       PROGRAM-ID. TEST1.
      *
      *
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-PC.
       OBJECT-COMPUTER. IBM-PC.
      *
      *
       DATA DIVISION.
      *
       WORKING-STORAGE SECTION.
      *
      * BTRIEVE OP CODES
      *
       01 B-OPEN                      PIC 9(4) COMP-5 VALUE 0.
       01 B-INSERT                    PIC 9(4) COMP-5 VALUE 2.
       01 B-GETFIRST                  PIC 9(4) COMP-5 VALUE 12.
       01 B-UPDATE                    PIC 9(4) COMP-5 VALUE 3.
       01 B-CLOSE                     PIC 9(4) COMP-5 VALUE 1.
      *
       01 B-STATUS                    PIC 9(4) COMP-5 VALUE 0.
       01 KEY-NUMBER                  PIC 9(4) COMP-5 VALUE 0.
```

```
      01 BUF-LEN                      PIC 9(4) COMP-5 VALUE 0.
      01 FILE-NAME                    PIC X(13) VALUE SPACES.
      01 POSITION-BLOCK               PIC X(128) VALUE SPACES.
       01 DATA-BUFFER.
         02 DECIMAL-FIELD             PIC 9(4) COMP-3 VALUE 0.
         02 STRING-FIELD              PIC X(36) VALUE SPACES.
       01 DSP-STATUS                    PIC 9(5) COMP-5.
      *
      *
      *
       PROCEDURE DIVISION.
       BEGIN.
      *
      * Open TEST.BTR
      *
      *
           MOVE 0   TO BUF-LEN.
           MOVE 0   TO KEY-NUMBER.
           MOVE 'TEST.BTR ' TO FILE-NAME.
          CALL "_BTRV" USING B-OPEN, B-STATUS, POSITION-BLOCK,
                           DATA-BUFFER, BUF-LEN, FILE-NAME,
KEY-NUMBER.
           IF B-STATUS NOT = 0
             DISPLAY 'Error opening file. Status= ' B-STATUS
           ELSE
             DISPLAY 'File ' FILE-NAME ' successfully opened'
           END-IF.
      *
      * Insert into TEST.BTR
      *
      *
```

Example A-3 Btrsamp.cbl *continued*

```
           MOVE 1    TO DECIMAL-FIELD.
           MOVE 'Record 1' TO STRING-FIELD.
           MOVE 40   TO BUF-LEN.
           MOVE 0    TO KEY-NUMBER
         CALL "_BTRV" USING B-INSERT, B-STATUS, POSITION-BLOCK,
                            DATA-BUFFER, BUF-LEN, FILE-NAME,
KEY-NUMBER.
           IF B-STATUS NOT = 0
             DISPLAY 'Error inserting into file. Status= ' B-
STATUS
           ELSE
             DISPLAY 'Inserted:  ' DECIMAL-FIELD STRING-FIELD
           END-IF.
       *
       * GetFirst
       *
       *
           MOVE 40   TO BUF-LEN.
           MOVE 0    TO KEY-NUMBER
           CALL "_BTRV" USING B-GETFIRST, B-STATUS, POSITION-
BLOCK,
                            DATA-BUFFER, BUF-LEN, FILE-NAME,
KEY-NUMBER.
           IF B-STATUS NOT = 0
             DISPLAY 'Error Getting first record. Status= ' B-
STATUS
           ELSE
             DISPLAY 'Retrieved:  ' DECIMAL-FIELD STRING-FIELD
           END-IF.
       *
       * Update into TEST.BTR
```

```
       *
       *
           MOVE 2    TO DECIMAL-FIELD.
           MOVE 'Record 2' TO STRING-FIELD.
           MOVE 40   TO BUF-LEN.
           MOVE 0    TO KEY-NUMBER
        CALL "_BTRV" USING B-UPDATE, B-STATUS, POSITION-BLOCK,
                           DATA-BUFFER, BUF-LEN, FILE-NAME,
KEY-NUMBER.
           IF B-STATUS NOT = 0
             DISPLAY 'Error updating file. Status= ' B-STATUS
           ELSE
            DISPLAY 'Updated to:  ' DECIMAL-FIELD STRING-FIELD
           END-IF.
       *
       * Close TEST.BTR
       *
       *
           MOVE 0    TO BUF-LEN.
           MOVE 0    TO KEY-NUMBER
        CALL "_BTRV" USING B-CLOSE, B-STATUS, POSITION-BLOCK,
                           DATA-BUFFER, BUF-LEN, FILE-NAME,
KEY-NUMBER.
           IF B-STATUS NOT = 0
             DISPLAY 'Error closing file. Status= ' B-STATUS
           ELSE
             DISPLAY 'Successfully closed TEST.BTR'
           END-IF.


           STOP RUN.
```

Delphi

This section discusses the following topics:

- ◆ [Program Example](#)
- ◆ [Compiling, Linking, and Running the Program Example](#)

Program Example

The following example program, which is included on your distribution media, shows how to perform several of the more common Btrieve operations, and it performs those operations in the order required by the MicroKernel's dependencies (for example, you must open a file before performing I/O to it).

Example A-4   Btrsam32.pas

```
{********************************************************
**
**   Copyright 1998 Pervasive Software Inc. All Rights Reserved
**
********************************************************}
{********************************************************
   BTRSAM32.DPR
      This is a simple sample designed to allow you to confirm
your ability to compile, link, and execute a Btrieve
application for your target 32-bit environment using your
compiler tools.

     This program demonstrates the Delphi interface for Btrieve
on 32-Bit MS Windows NT and Windows 95, for Delphi 2.0 and 3.0.
```

```
     This program does the following operations on the sample
file:
    - gets the Microkernel Database Engine version using BTRVID
    - opens sample.btr
    - gets a record on a known value of Key 0
    - displays the retrieved record
    - performs a stat operation
    - creates an empty 'clone' of sample.btr and opens it
    - performs a 'Get Next Extended' operation to extract a
subset of the records in sample.btr
    - inserts those records into the cloned file
    - closes both files


     IMPORTANT:
    You must specify the complete path to the directory that
contains the sample Btrieve data file, 'sample.btr'.  See
IMPORTANT, below. Delphi 2.0/3.0 Btrieve projects must be
compiled after selecting the following from the Delphi project
environment pull-down menus:

        PROJECT
           OPTIONS...
              COMPILER
                 CODE GENERATION
                    ALIGNED RECORD FIELDS ( de-select this )

```

```
        If you don't do this step, when the record is printed
out, it will seem 'jumbled' because the record structure is
not byte-packed. You may, instead, use the (*A-*) compiler
directive, or declare all records as "packed," as shown below.
For more information, see the Delphi documentation.

      PROJECT FILES:
        - btr32.dpr       Borland project file
        - btr32.dof       Borland project file
        - btrsam32.dfm    Borland project file
        - btrsam32.pas    Source code for the simple sample
        - btrapi32.pas    Delphi interface to Btrieve
        - btrconst.pas    Btrieve constants file

********************************************************}
unit btrsam32;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  StdCtrls, BtrConst, BtrAPI32;

{***********************************************************
   Program Constants
***********************************************************}
const
  { program constants }
  MY_THREAD_ID       = 50;
  EXIT_WITH_ERROR    = 1;
```

```
  VERSION_OFFSET        = 0;
  REVISION_OFFSET       = 2;
  PLATFORM_ID_OFFSET    = 4;

  {***********************************************************
     IMPORTANT: You should modify the following to specify the
  complete path to 'sample.btr' for your environment.
  ***********************************************************}
  FILE_1                = 'c:\pvsw\samples\sample.btr';
  FILE_2                = 'c:\pvsw\samples\sample2.btr';

{*************************************************************
  Record type definitions for Version operation
*************************************************************}
type
  CLIENT_ID = packed record
    networkandnode : array[1..12] of char;
    applicationID  : array[1..3] of char;
    threadID       : smallint;
  end;

  VERSION_STRUCT = packed record
    version   : smallint;
    revision  : smallint;
    MKDEId    : char;
  end;

{*************************************************************
  Definition of record from 'sample.btr'
*************************************************************}
```

```
  {* Use 'zero-based' arrays of char for writeln()
compatibility *}
  PERSON_STRUCT = packed record
    ID          : longint;
    FirstName   : array[0..15] of char;
    LastName    : array[0..25] of char;
    Street      : array[0..30] of char;
    City        : array[0..30] of char;
    State       : array[0..2]  of char;
    Zip         : array[0..10] of char;
    Country     : array[0..20] of char;
    Phone       : array[0..13] of char;
  end;

{************************************************************
  Record type definitions for Stat and Create operations
************************************************************}
  FILE_SPECS = packed record
    recLength   : smallint;
    pageSize    : smallint;
    indexCount  : smallint;
    reserved    : array[0..3] of char;
    flags       : smallint;
    dupPointers : byte;
    notUsed     : byte;
    allocations : smallint;
  end;

  KEY_SPECS = packed record
    position : smallint;
    length : smallint;
```

```
    flags : smallint;
    reserved : array [0..3] of char;
    keyType : char;
    nullChar : char;
    notUsed : array[0..1] of char;
    manualKeyNumber : byte;
    acsNumber : byte;
  end;

  FILE_CREATE_BUF = packed record
    fileSpecs : FILE_SPECS;
    keySpecs  : array[0..4] of KEY_SPECS;
  end;

{************************************************************
  Record type definitions for Get Next Extended operation
 ************************************************************}

  GNE_HEADER = packed record
    descriptionLen  : smallint;
    currencyConst   : array[0..1] of char;
    rejectCount     : smallint;
    numberTerms     : smallint;
  end;

  TERM_HEADER = packed record
    fieldType       : byte;
    fieldLen        : smallint;
    fieldOffset     : smallint;
    comparisonCode  : byte;
    connector       : byte;
```

```
    value             : array[0..2] of char;
  end;

  RETRIEVAL_HEADER = packed record
    maxRecsToRetrieve   : smallint;
    noFieldsToRetrieve  : smallint;
  end;

  FIELD_RETRIEVAL_HEADER = packed record
    fieldLen    : smallint;
    fieldOffset : smallint;
  end;

  PRE_GNE_BUFFER = packed record
    gneHeader : GNE_HEADER;
    term1     : TERM_HEADER;
    term2     : TERM_HEADER;
    retrieval : RETRIEVAL_HEADER;
    recordRet : FIELD_RETRIEVAL_HEADER;
  end;

  RETURNED_REC = packed record
    recLen        : smallint;
    recPos        : longint;
    personRecord  : PERSON_STRUCT;
  end;

  POST_GNE_BUFFER = packed record
    numReturned : smallint;
    recs        : packed array[0..19] of RETURNED_REC;
  end;
```

```
  GNE_BUFFER_PTR = ^GNE_BUFFER;
  GNE_BUFFER = packed record
  case byte of
    1 : (preBuf  : PRE_GNE_BUFFER);
    2 : (postBuf : POST_GNE_BUFFER);
  end;

{***********************************************************
  Delphi-generated form definition
 ***********************************************************}
  TForm1 = class(TForm)
    RunButton: TButton;
    ExitButton: TButton;
    ListBox1: TListBox;
    procedure FormCreate(Sender: TObject);
    procedure ExitButtonClick(Sender: TObject);
    procedure RunButtonClick(Sender: TObject);
  private
  { Private declarations }
    ArrowCursor,
    WaitCursor:   HCursor;
    status:       smallint;
    bufferLength: smallint;
    personRecord: PERSON_STRUCT;
    recordsRead:  longint;
    procedure RunTest;
  public
    { Public declarations }
  end;
```

Example A-4   Btrsam32.pas *continued*

```
var
  Form1: TForm1;

{********************************************************
  Program starts here
********************************************************}
implementation

{$R *.DFM}

{********************************************************
   Program Variables
********************************************************}
var
  { Btrieve function parameters }
  posBlock1     : string[128];
  posBlock2     : string[128];
  dataBuffer    : array[0..255] of char;
  dataLen       : word;
  keyBuf1       : string[255];
  keyBuf2       : string[255];
  keyNum        : smallint;

  btrieveLoaded : boolean;
  personID      : longint;
  file1Open     : boolean;
  file2Open     : boolean;
  status        : smallint;
  getStatus     : smallint;
  i             : smallint;
  posCtr        : smallint;
```

```
  client        : CLIENT_ID;
  versionBuffer : array[1..3] of VERSION_STRUCT;
  fileCreateBuf : FILE_CREATE_BUF;
  gneBuffer     : GNE_BUFFER_PTR;
  personRecord  : PERSON_STRUCT;

{***********************************************************
   A helper procedure to write to the ListBox
***********************************************************}
procedure WritelnLB( LB: TListBox; Str: String);
begin
  LB.Items.Add(Str);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ArrowCursor   :=  LoadCursor(0, IDC_ARROW);
  WaitCursor    :=  LoadCursor(0, IDC_WAIT);
end;

{***********************************************************
   This is the 'main' procedure of the sample
***********************************************************}
procedure TForm1.RunTest;
begin
  ListBox1.Clear;
  WritelnLB( ListBox1, 'Test started ...' );

  { initialize variables }
  btrieveLoaded := FALSE;
  file1Open := FALSE;
```

```
   file2Open := FALSE;
   keyNum := 0;
   status := B_NO_ERROR;
   getStatus := B_NO_ERROR;

   { set up the Client ID }
   fillchar(client.networkAndNode,
sizeof(client.networkAndNode), #0);
   client.applicationID := 'MT' + #0;  { must be greater than
"AA" }
   client.threadID := MY_THREAD_ID;

   fillchar(versionBuffer, sizeof(versionBuffer), #0);
   dataLen := sizeof(versionBuffer);

   status := BTRVID(
               B_VERSION,
               posBlock1,
               versionBuffer,
               dataLen,
               keyBuf1[1],
               keyNum,
               client);

   if status = B_NO_ERROR then begin
     writelnLB( ListBox1, 'Btrieve Versions returned are:' );
     for i := 1 to 3 do begin
       with versionBuffer[i] do begin
         if (version > 0) then begin
           writelnLB(ListBox1, intToStr(version) + '.' +
                       intToStr(revision) + ' ' + MKDEId);
```

```
        end
      end
    end;
    btrieveLoaded := TRUE;
  end else begin
    writelnLB(ListBox1, 'Btrieve B_VERSION status = ' +
intToStr(status));
    if status <> B_RECORD_MANAGER_INACTIVE then begin
      btrieveLoaded := TRUE;
    end
  end;

  { open sample.btr }
  if status = B_NO_ERROR then begin
    fillchar(dataBuffer, sizeof(dataBuffer), #0);
    fillchar(keyBuf1, sizeof(keyBuf1), #0);
    keyNum := 0;
    dataLen := 0;

    keyBuf1 := FILE_1 + #0;
    keyBuf2 := FILE_2 + #0;

    status := BTRVID(
                 B_OPEN,
                 posBlock1,
                 dataBuffer,
                 dataLen,
                 keyBuf1[1],
                 keyNum,
                 client);
```

```
    writelnLB(ListBox1, 'Btrieve B_OPEN status = ' +
intToStr(status));
    if status = B_NO_ERROR then begin
      file1Open := TRUE;
    end
  end;

  {* get the record using key 0 = a known value using
B_GET_EQUAL *}
  if status = B_NO_ERROR then begin
    fillchar(personRecord, sizeof(personRecord), #0);
    dataLen := sizeof(personRecord);
   personID := 263512477;  {* this is really a social security
number *}

    status := BTRVID(
                B_GET_EQUAL,
                posBlock1,
                personRecord,
                dataLen,
                personID,
                keyNum,
                client);

    writelnLB(ListBox1, 'Btrieve B_GET_EQUAL status = ' +
intToStr(status));
    if status = B_NO_ERROR then with personRecord do begin
      writelnLB(ListBox1, '');
      writelnLB(ListBox1, 'Selected fields from the retrieved
record are:');
      writelnLB(ListBox1, 'ID:       ' + intToStr(ID));
```

Example A-4   Btrsam32.pas *continued*

```
      writelnLB(ListBox1, 'Name:     ' + FirstName + ' ' +
                            LastName);
      writelnLB(ListBox1, 'Street:  ' + Street);
      writelnLB(ListBox1, 'City:    ' + City);
      writelnLB(ListBox1, 'State:   ' + State);
      writelnLB(ListBox1, 'Zip:     ' + Zip);
      writelnLB(ListBox1, 'Country: ' + Country);
      writelnLB(ListBox1, 'Phone:   ' + Phone);
      writelnLB(ListBox1, '');
    end;
  end;

  { perform a stat operation to populate the create buffer }
  fillchar(fileCreateBuf, sizeof(fileCreateBuf), #0);
  dataLen := sizeof(fileCreateBuf);
  keyNum  := -1;
  status := BTRVID(B_STAT,
                posBlock1,
                fileCreateBuf,
                dataLen,
                keyBuf1[1],
                keyNum,
                client);

  if (status = B_NO_ERROR) then begin
    { create and open sample2.btr }
    keyNum := 0;
    dataLen := sizeof(fileCreateBuf);
    status := BTRVID(B_CREATE,
                  posBlock2,
                  fileCreateBuf,
```

```
                    dataLen,
                    keyBuf2[1],
                    keyNum,
                    client);

    writelnLB(ListBox1, 'Btrieve B_CREATE status = ' +
intToStr(status));
  end;

  if (status = B_NO_ERROR) then begin
    keyNum  := 0;
    dataLen := 0;

    status := BTRVID(
                 B_OPEN,
                 posBlock2,
                 dataBuffer,
                 dataLen,
                 keyBuf2[1],
                 keyNum,
                 client);
    writelnLB(ListBox1, 'Btrieve B_OPEN status = ' +
intToStr(status));
    if (status = B_NO_ERROR) then begin
      file2Open := TRUE;
    end;
  end;

  { now extract data from the original file, insert into new
one }
  if (status = B_NO_ERROR) then begin
```

```
    { getFirst to establish currency }
    keyNum := 2; { STATE-CITY index }
    fillchar(personRecord, sizeof(personRecord), #0);
    fillchar(keyBuf1, sizeof(keyBuf1), #0);
    dataLen := sizeof(personRecord);

    getStatus := BTRVID(
                    B_GET_FIRST,
                    posBlock1,
                    personRecord,
                    dataLen,
                    keyBuf1[1],
                    keyNum,
                    client);

    writelnLB(ListBox1, 'Btrieve B_GET_FIRST status = ' +
intToStr(GETstatus));
    writelnLB(ListBox1, '');
  end;

  { Allocate memory on heap }
  gneBuffer := new(GNE_BUFFER_PTR);
  fillchar(gneBuffer^, sizeof(GNE_BUFFER), #0);

  strPCopy(gneBuffer^.preBuf.gneHeader.currencyConst, 'UC');
  while (getStatus = B_NO_ERROR) do begin
    gneBuffer^.preBuf.gneHeader.rejectCount := 0;
    gneBuffer^.preBuf.gneHeader.numberTerms := 2;
    posCtr := sizeof(GNE_HEADER);

    { fill in the first condition }
```

```
     gneBuffer^.preBuf.term1.fieldType := 11;
     gneBuffer^.preBuf.term1.fieldLen := 3;
     gneBuffer^.preBuf.term1.fieldOffset := 108;
     gneBuffer^.preBuf.term1.comparisonCode := 1;
     gneBuffer^.preBuf.term1.connector := 2;

     strPCopy(gneBuffer^.preBuf.term1.value, 'TX');
     inc(posCtr, (sizeof(TERM_HEADER)));

     { fill in the second condition }
     gneBuffer^.preBuf.term2.fieldType := 11;
     gneBuffer^.preBuf.term2.fieldLen := 3;
     gneBuffer^.preBuf.term2.fieldOffset := 108;
     gneBuffer^.preBuf.term2.comparisonCode := 1;
     gneBuffer^.preBuf.term2.connector := 0;
     strPCopy(gneBuffer^.preBuf.term2.value, 'CA');
     inc(posCtr, sizeof(TERM_HEADER));

     { fill in the projection header to retrieve whole record }
     gneBuffer^.preBuf.retrieval.maxRecsToRetrieve := 20;
     gneBuffer^.preBuf.retrieval.noFieldsToRetrieve := 1;
     inc(posCtr, sizeof(RETRIEVAL_HEADER));
     gneBuffer^.preBuf.recordRet.fieldLen :=
sizeof(PERSON_STRUCT);
     gneBuffer^.preBuf.recordRet.fieldOffset := 0;
     inc(posCtr, sizeof(FIELD_RETRIEVAL_HEADER));
     gneBuffer^.preBuf.gneHeader.descriptionLen := posCtr;

     dataLen := sizeof(GNE_BUFFER);
     getStatus := BTRVID(
                    B_GET_NEXT_EXTENDED,
```

```
                    posBlock1,
                    gneBuffer^,
                    dataLen,
                    keyBuf1,
                    keyNum,
                    client);

    writelnLB(ListBox1, 'Btrieve B_GET_NEXT_EXTENDED status =
' + intToStr(getStatus));

    { Get Next Extended can reach end of file and still return
some records }
    if ((getStatus = B_NO_ERROR) or (getStatus =
B_END_OF_FILE)) then begin
       writelnLB(ListBox1, 'GetNextExtended returned ' +
                 intToStr(gneBuffer^.postBuf.numReturned) + '
records.');
       for i := 0 to gneBuffer^.postBuf.numReturned - 1 do begin
          dataLen := sizeof(PERSON_STRUCT);
          personRecord :=
gneBuffer^.postBuf.recs[i].personRecord;
          status := BTRVID(
                    B_INSERT,
                    posBlock2,
                    personRecord,
                    dataLen,
                    keyBuf2,
                    -1,   { no currency change }
                    client);
          if (status <> B_NO_ERROR) then begin
```

```
          writelnLB(ListBox1, 'Btrieve B_INSERT status = ' +
intToStr(status));
          break;
        end;
      end;

      writelnLB(ListBox1, 'Inserted ' +
intToStr(gneBuffer^.postBuf.numReturned) +
                ' records in new file, status = ' +
intToStr(status));
      writelnLB(ListBox1, '');
    end;
    fillchar(gneBuffer^, sizeof(GNE_BUFFER), #0);
    gneBuffer^.preBuf.gneHeader.currencyConst := 'EG';
  end;
  dispose(gneBuffer);

  { close open files }
  keyNum := 0;
  if file1Open = TRUE then begin
    dataLen := 0;

    status := BTRVID(
                B_CLOSE,
                posBlock1,
                dataBuffer,
                dataLen,
                keyBuf1[1],
                keyNum,
                client);
```

```
     writelnLB(ListBox1, 'Btrieve B_CLOSE status (sample.btr)
= ' + intToStr(status));
   end;

   if file2Open = TRUE then begin
     dataLen := 0;

     status := BTRVID(
                  B_CLOSE,
                  posBlock2,
                  dataBuffer,
                  dataLen,
                  keyBuf2[1],
                  keyNum,
                  client);

    writelnLB(ListBox1, 'Btrieve B_CLOSE status (sample2.btr)
= ' + intToStr(status));
   end;

   { FREE RESOURCES }
   dataLen := 0;
   status := BTRVID( B_STOP, posBlock1, DataBuffer,
                dataLen, keyBuf1[1], 0, client );
   WritelnLB(ListBox1, 'Btrieve B_STOP status = ' +
intToStr(status) );

end;

procedure TForm1.ExitButtonClick(Sender: TObject);
begin
```

Example A-4   Btrsam32.pas *continued*

```
   Close;
end;

procedure TForm1.RunButtonClick(Sender: TObject);
begin
  SetCursor(WaitCursor);
  RunTest;
  SetCursor(ArrowCursor);
end;

end.
```

Compiling, Linking, and Running the Program Example

➤ **In Delphi 3, to compile, link, and run the program example:**

   1. Choose **New Application** from the **File** menu.

   2. Choose **Open** from the **File** menu and open the B32.dpr project file in the \Intf\Delphi directory.

   3. In the BtrSam32.pas file, edit the paths to the Sample.btr and Sample2.btr files as appropriate.

   4. Click the **Run** button on the toolbar.

   Delphi compiles and links the program example.

   5. Click the **Run Test** button in the Btrieve Sample Application window.

   Delphi runs the program example.

➤ **In Delphi 1, to compile, link, and run the program example:**

1. Choose **Open Project** from the **File** menu and open the B16.dpr project file in the \Intf\Delphi directory.

2. In the BtrSam16.pas file, edit the paths to the Sample.btr and Sample2.btr files as appropriate.

3. Click the **Run** button on the toolbar.

   Delphi compiles and links the program example.

4. Click the **Run Test** button in the Btrieve Sample Application window.

   Delphi runs the program example.

Pascal

This section discusses the following topics:

- ◆

- ◆

- ◆

- ◆

Source Modules

The Pascal interface is comprised of the following source modules:

- ◆ BtrApiD.PAS—Btrieve functions interface unit for 16-bit DOS.

- ◆ BtrApiW.PAS—Btrieve functions interface unit for Win16.

- ◆ BtrConst.PAS—Common Btrieve constants unit.

- ◆ BtrSampD.PAS—Sample Btrieve program for 16-bit DOS.

- ◆ BtrSampW.PAS—Sample Btrieve program for Win16.

BtrApiD.PAS and BtrApiW.PAS

The files BtrApiD.PAS and BtrApiW.PAS contain the source code implementation of the Pascal application interface for 16-bit DOS and Win16. These files provide support for applications calling Btrieve functions.

In order for Turbo Pascal to properly compile and link the Btrieve interface with the other modules in your application, you can compile BtrApiD.PAS or BtrApiW.PAS to create a Turbo Pascal unit which you then list in the `uses` clause of your application's source code.

BtrConst.PAS

The file BtrConst.PAS contains useful constants specific to Btrieve. These constants can help you standardize references to Btrieve operation codes, status codes, file specification flags, key specification flags, and many more items.

To use BtrConst.PAS, you can compile it to create a Turbo Pascal unit which you then list in the `uses` clause of your application's source code.

You can use the Pascal application interface without taking advantage of BtrConst.PAS; however, using the file may simplify your programming effort.

BtrSampD.PAS and BtrSampW.PAS

The source files BtrSampD.PAS and BtrSampW.PAS are sample Btrieve programs that you can compile, link, and run.

Compiling and Linking Using the Interface

This section provides instructions for some common development environments. If your development environment is not discussed, you can adapt these instructions.

➤ **Using Borland Pascal 7 for Windows, to compile, link, and run the program example:**

   1. Choose **Open** from the **File** menu and open the BtrSampW.pas file.

   2. In the BtrSampW.pas file, edit the paths to the Sample.btr and Sample2.btr files as appropriate.

   3. Choose **Make** from the **Compile** menu. Click OK.

   Borland Pascal compiles and links the program example.

   4. Choose **Run** from the **Run** menu.

   Borland Pascal runs the program example.

➤ **Using Borland Pascal 7 for DOS, to compile, link, and run the program example:**

   1. Choose **Open** from the **File** menu and open the following files:

   - \Intf\Pascal\BtrConst.pas

   - \Intf\Pascal\BtrApiD.pas

   - \Intf\Pascal\BtrSampD.pas

   2. In the BtrSampD.pas file, edit the paths to the Sample.btr and Sample2.btr files as appropriate.

   3. Choose **Compiler** from the **Options** menu and set the Conditional Defines to BTI_DOS.

4. Choose **Directories** from the **Options** menu and set EXE and TPU to the \Intf\Pascal directory.

5. In the BtrConst.pas file, choose **Make** from the **Compile** menu.

6. In the BtrApiD.pas file, choose **Make** from the **Compile** menu.

7. In the BtrSampD.pas file, choose **Make** from the **Compile** menu.

8. In a DOS prompt, run BtrSampD.exe.

Note

The program example uses the BtrvID call; therefore, you must load the Btrieve DOS Requester with the /T:1 flag. Also, you must load the appropriate Requester (BRequest for NetWare servers and BReqNT for Windows NT servers).

➤ **Using Borland Turbo Pascal 1.5 for Windows, to compile, link, and run the program example:**

1. Choose **Open** from the **File** menu and open the following files:

   • \Intf\Pascal\BtrConst.pas

   • \Intf\Pascal\BtrApiW.pas

   • \Intf\Pascal\BtrSampW.pas

2. In the BtrSampW.pas file, edit the paths to the Sample.btr and Sample2.btr files as appropriate.

3. In the BtrConst.pas file, choose **Make** from the **Compile** menu.

4. In the BtrApiW.pas file, choose **Make** from the **Compile** menu.

5. In the BtrSampW.pas file, choose **Make** from the **Compile** menu.

6. Choose **Run** from the **Run** menu.

   Borland Turbo Pascal runs the program example.

Program Example

The following example program, which is included on your distribution media, shows how to perform several of the more common Btrieve operations, and it performs those operations in the order required by the MicroKernel's dependencies (for example, you must open a file before performing I/O to it).

Example A-5   Btrsampw.pas

```
{********************************************************
**
**   Copyright 1998 Pervasive Software Inc. All Rights Reserved
**
********************************************************}
{********************************************************
{
  BTRSAMPW.PAS
    This program demonstrates the Btrieve Interface for MS
Windows using Borland Pascal version 7.0 or Turbo Pascal for
Windows version 1.5.

    This program does the following operations on the sample
file:
    - gets the Microkernel Database Engine version using BTRVID
    - opens sample.btr
    - gets a record on a known value of Key 0
    - displays the retrieved record
    - performs a stat operation
    - creates an empty 'clone' of sample.btr and opens it
    - performs a 'Get Next Extended' operation to extract a
subset of the records in sample.btr
    - inserts those records into the cloned file
    - closes both files
```

```
    IMPORTANT:
   You must specify the complete path to the directory that
contains the sample Btrieve data file, 'sample.btr'.  See
IMPORTANT, below.
***********************************************************}
program btrsampw;

uses
  WinCrt,     { text mode I/O library for Windows }
  Strings,    { Pascal System functions }
  btrapiw,    { btrieve interface unit }
  btrconst;   { Btrieve Constants Unit }

const
  {***********************************************************
     IMPORTANT: You should modify the following to specify the
 complete path to 'sample.btr' for your environment.
 ***********************************************************}
  FILE1_NAME            = 'c:\pvsw\samples\sample.btr' + #0;
  FILE2_NAME            = 'c:\pvsw\samples\sample2.btr' + #0;

  { program constants }
  MY_THREAD_ID        = 50;
  EXIT_WITH_ERROR     = 1;
  VERSION_OFFSET      = 0;
  REVISION_OFFSET     = 2;
  PLATFORM_ID_OFFSET  = 4;

{***********************************************************
  Record type definitions for Version operation
***********************************************************}
```

Example A-5   Btrsampw.pas *continued*

```
type
  CLIENT_ID = packed record
    networkandnode : array[0..11] of char;
    applicationID  : array[0..2] of char;
    threadID       : integer;
  end;

  VERSION_STRUCT = packed record
    version  : integer;
    revision : integer;
    MKDEId   : char;
  end;

{**********************************************************
  Definition of record from 'sample.btr'
**********************************************************}

  {* Use 'zero-based' arrays of char for writeln()
compatibility *}
  PERSON_STRUCT = packed record
    ID         : longint;
    FirstName  : array[0..15] of char;
    LastName   : array[0..25] of char;
    Street     : array[0..30] of char;
    City       : array[0..30] of char;
    State      : array[0..2]  of char;
    Zip        : array[0..10] of char;
    Country    : array[0..20] of char;
    Phone      : array[0..13] of char;
  end;
```

Example A-5   Btrsampw.pas *continued*

```
{***********************************************************
  Record type definitions for Stat and Create operations
 ***********************************************************}
  FILE_SPECS = packed record
    recLength   : integer;
    pageSize    : integer;
    indexCount  : integer;
    reserved    : array[0..3] of char;
    flags       : integer;
    dupPointers : byte;
    notUsed     : byte;
    allocations : integer;
  end;

  KEY_SPECS = packed record
    position : integer;
    length : integer;
    flags : integer;
    reserved : array [0..3] of char;
    keyType : char;
    nullChar : char;
    notUsed : array[0..1] of char;
    manualKeyNumber : byte;
    acsNumber : byte;
  end;

  FILE_CREATE_BUF = packed record
    fileSpecs : FILE_SPECS;
    keySpecs  : array[0..4] of KEY_SPECS;
  end;
```

Example A-5   Btrsampw.pas *continued*

```
{************************************************************
  Record type definitions for Get Next Extended operation
************************************************************}

  GNE_HEADER = packed record
    descriptionLen  : integer;
    currencyConst   : array[0..1] of char;
    rejectCount     : integer;
    numberTerms     : integer;
  end;

  TERM_HEADER = packed record
    fieldType       : byte;
    fieldLen        : integer;
    fieldOffset     : integer;
    comparisonCode  : byte;
    connector       : byte;
    value           : array[0..2] of char;
  end;

  RETRIEVAL_HEADER = packed record
    maxRecsToRetrieve   : integer;
    noFieldsToRetrieve  : integer;
  end;

  FIELD_RETRIEVAL_HEADER = packed record
    fieldLen    : integer;
    fieldOffset : integer;
  end;

  PRE_GNE_BUFFER = packed record
```

```
   gneHeader : GNE_HEADER;
   term1     : TERM_HEADER;
   term2     : TERM_HEADER;
   retrieval : RETRIEVAL_HEADER;
   recordRet : FIELD_RETRIEVAL_HEADER;
 end;

 RETURNED_REC = packed record
   recLen        : integer;
   recPos        : longint;
   personRecord  : PERSON_STRUCT;
 end;

 POST_GNE_BUFFER = packed record
   numReturned : integer;
   recs        : packed array[0..19] of RETURNED_REC;
 end;

 GNE_BUFFER_PTR = ^GNE_BUFFER;
 GNE_BUFFER = packed record
 case byte of
   1 : (preBuf  : PRE_GNE_BUFFER);
   2 : (postBuf : POST_GNE_BUFFER);
 end;

{***********************************************************
 Variables
***********************************************************}
var
 { Btrieve function parameters }
 posBlock1      : string[128];
```

Example A-5   Btrsampw.pas *continued*

```
   posBlock2      : string[128];
   dataBuffer     : array[0..255] of char;
   dataLen        : word;
   keyBuf1        : string[255];
   keyBuf2        : string[255];
   keyNum         : integer;

   btrieveLoaded : boolean;
   personID      : longint;
   file1Open     : boolean;
   file2Open     : boolean;
   status        : integer;
   getStatus     : integer;
   i             : integer;
   posCtr        : integer;

   client        : CLIENT_ID;
   versionBuffer : array[1..3] of VERSION_STRUCT;
   fileCreateBuf : FILE_CREATE_BUF;
   gneBuffer     : GNE_BUFFER_PTR;
   personRecord  : PERSON_STRUCT;

{***********************************************************
   Program starts here
 ***********************************************************}
begin { btrsamp }
   { initialize variables }
   btrieveLoaded := FALSE;
   file1Open := FALSE;
   file2Open := FALSE;
   keyNum := 0;
```

```
  status := B_NO_ERROR;
  getStatus := B_NO_ERROR;

  writeln;
  writeln(`************ Btrieve Pascal Interface for Windows
Demo ************');
  writeln;

  { set up the Client ID }
  fillchar(client.networkAndNode,
sizeof(client.networkAndNode), #0);

{$ifdef ver70} {Note: Delphi 1.0 is ver80}
  client.applicationID := 'MT' + #0;     { must be greater than
"AA" }
{$else}
  strpcopy(client.applicationID, 'MT'); { must be greater than
"AA" }
  strcat(client.applicationID, #0);
{$endif}

  client.threadID := MY_THREAD_ID;

  fillchar(versionBuffer, sizeof(versionBuffer), #0);
  dataLen := sizeof(versionBuffer);

  status := BTRVID(
              B_VERSION,
              posBlock1,
              versionBuffer,
              dataLen,
```

```
                   keyBuf1[1],
                   keyNum,
                   client);

    if status = B_NO_ERROR then begin
      writeln('Btrieve Versions returned are:');
      for i := 1 to 3 do begin
        with versionBuffer[i] do begin
          if (version > 0) then begin
            writeln(version, '.', revision, ' ', MKDEId);
          end
        end
      end;
      btrieveLoaded := TRUE;
    end else begin
      writeln('Btrieve B_VERSION status = ', status);
      if status <> B_RECORD_MANAGER_INACTIVE then begin
        btrieveLoaded := TRUE;
      end
    end;

    {* open sample.btr *}
    if status = B_NO_ERROR then begin
      fillchar(dataBuffer, sizeof(dataBuffer), #0);
      fillchar(keyBuf1, sizeof(keyBuf1), #0);
      keyNum := 0;
      dataLen := 0;

      keyBuf1 := FILE1_NAME;
      keyBuf2 := FILE2_NAME;
```

```
      status := BTRVID(
                   B_OPEN,
                   posBlock1,
                   dataBuffer,
                   dataLen,
                   keyBuf1[1],
                   keyNum,
                   client);

    writeln('Btrieve B_OPEN status = ', status);
    if status = B_NO_ERROR then begin
      file1Open := TRUE;
    end
  end;

  {* get the record using key 0 = a known value using
B_GET_EQUAL *}
  if status = B_NO_ERROR then begin
    fillchar(personRecord, sizeof(personRecord), #0);
    dataLen := sizeof(personRecord);
   personID := 263512477;  {* this is really a social security
number *}

    status := BTRVID(
                 B_GET_EQUAL,
                 posBlock1,
                 personRecord,
                 dataLen,
                 personID,
                 keyNum,
                 client);
```

```
    writeln('Btrieve B_GET_EQUAL status = ', status);
    if status = B_NO_ERROR then with personRecord do begin
      writeln;
      writeln('Selected fields from the retrieved record
are:');
      writeln('ID:       ', ID);
      writeln('Name:     ', FirstName, ' ',
                            LastName);
      writeln('Street: ', Street);
      writeln('City:     ', City);
      writeln('State:    ', State);
      writeln('Zip:      ', Zip);
      writeln('Country: ', Country);
      writeln('Phone:    ', Phone);
      writeln;
    end;
  end;

  { perform a stat operation to populate the create buffer }
  fillchar(fileCreateBuf, sizeof(fileCreateBuf), #0);
  dataLen := sizeof(fileCreateBuf);
  keyNum  := -1;
  status := BTRVID(B_STAT,
                   posBlock1,
                   fileCreateBuf,
                   dataLen,
                   keyBuf1[1],
                   keyNum,
                   client);
```

```
  if (status = B_NO_ERROR) then begin
    { create and open sample2.btr }
    keyNum := 0;
    dataLen := sizeof(fileCreateBuf);
    status := BTRVID(B_CREATE,
                     posBlock2,
                     fileCreateBuf,
                     dataLen,
                     keyBuf2[1],
                     keyNum,
                     client);

    writeln('Btrieve B_CREATE status = ', status);
  end;

  if (status = B_NO_ERROR) then begin
    keyNum  := 0;
    dataLen := 0;

    status := BTRVID(
                     B_OPEN,
                     posBlock2,
                     dataBuffer,
                     dataLen,
                     keyBuf2[1],
                     keyNum,
                     client);
    writeln('Btrieve B_OPEN status (sample2.btr) = ', status);
    if (status = B_NO_ERROR) then begin
      file2Open := TRUE;
    end;
```

```
    end;

  { now extract data from the original file, insert into new
one }
  if (status = B_NO_ERROR) then begin
    { getFirst to establish currency }
    keyNum := 2; { STATE-CITY index }
    fillchar(personRecord, sizeof(personRecord), #0);
    fillchar(keyBuf1, sizeof(keyBuf1), #0);
    dataLen := sizeof(personRecord);

    getStatus := BTRVID(
                   B_GET_FIRST,
                   posBlock1,
                   personRecord,
                   dataLen,
                   keyBuf1[1],
                   keyNum,
                   client);

    writeln('Btrieve B_GET_FIRST status (sample.btr) = ',
getStatus);
    writeln;
  end;

  if maxavail < SizeOf(GNE_BUFFER) then begin
    writeln('Insufficient memory to allocate buffer');
    halt(EXIT_WITH_ERROR);
  end else begin
    { Allocate memory on heap }
    gneBuffer := new(GNE_BUFFER_PTR);
```

```
    end;
    fillchar(gneBuffer^, sizeof(GNE_BUFFER), #0);
    strPCopy(gneBuffer^.preBuf.gneHeader.currencyConst, 'UC');
    while (getStatus = B_NO_ERROR) do begin
      gneBuffer^.preBuf.gneHeader.rejectCount := 0;
      gneBuffer^.preBuf.gneHeader.numberTerms := 2;
      posCtr := sizeof(GNE_HEADER);

      { fill in the first condition }
      gneBuffer^.preBuf.term1.fieldType := 11;
      gneBuffer^.preBuf.term1.fieldLen := 3;
      gneBuffer^.preBuf.term1.fieldOffset := 108;
      gneBuffer^.preBuf.term1.comparisonCode := 1;
      gneBuffer^.preBuf.term1.connector := 2;

      strPCopy(gneBuffer^.preBuf.term1.value, 'TX');
      inc(posCtr, (sizeof(TERM_HEADER)));

      { fill in the second condition }
      gneBuffer^.preBuf.term2.fieldType := 11;
      gneBuffer^.preBuf.term2.fieldLen := 3;
      gneBuffer^.preBuf.term2.fieldOffset := 108;
      gneBuffer^.preBuf.term2.comparisonCode := 1;
      gneBuffer^.preBuf.term2.connector := 0;
      strPCopy(gneBuffer^.preBuf.term2.value, 'CA');
      inc(posCtr, sizeof(TERM_HEADER));

      { fill in the projection header to retrieve whole record }
      gneBuffer^.preBuf.retrieval.maxRecsToRetrieve := 20;
      gneBuffer^.preBuf.retrieval.noFieldsToRetrieve := 1;
      inc(posCtr, sizeof(RETRIEVAL_HEADER));
```

```
     gneBuffer^.preBuf.recordRet.fieldLen :=
sizeof(PERSON_STRUCT);
     gneBuffer^.preBuf.recordRet.fieldOffset := 0;
     inc(posCtr, sizeof(FIELD_RETRIEVAL_HEADER));
     gneBuffer^.preBuf.gneHeader.descriptionLen := posCtr;

     dataLen := sizeof(GNE_BUFFER);
     getStatus := BTRVID(
                    B_GET_NEXT_EXTENDED,
                    posBlock1,
                    gneBuffer^,
                    dataLen,
                    keyBuf1,
                    keyNum,
                    client);

      writeln('Btrieve B_GET_NEXT_EXTENDED status = ',
getStatus);

     { Get Next Extended can reach end of file and still return
some records }
     if ((getStatus = B_NO_ERROR) or (getStatus =
B_END_OF_FILE)) then begin
       writeln('GetNextExtended returned ',
gneBuffer^.postBuf.numReturned, ' records.');
       for i := 0 to gneBuffer^.postBuf.numReturned - 1 do begin

{$ifdef ver70}
         dataLen := sizeof(PERSON_STRUCT);
         personRecord :=
gneBuffer^.postBuf.recs[i].personRecord;
```

```
          status := BTRVID(
                     B_INSERT,
                     posBlock2,
                     personRecord,
                     dataLen,
                     keyBuf2,
                     -1,   { no currency change }
                     client);
        if (status <> B_NO_ERROR) then begin
          writeln('Btrieve B_INSERT status = ', status);
          break;
        end;

{$else} {Turbo Pascal for Windows 1.5 does not support 'break'}
        if (status = B_NO_ERROR) then begin
          dataLen := sizeof(PERSON_STRUCT);
          personRecord :=
gneBuffer^.postBuf.recs[i].personRecord;
          status := BTRVID(
                     B_INSERT,
                     posBlock2,
                     personRecord,
                     dataLen,
                     keyBuf2,
                     -1,   { no currency change }
                     client);
        end;
        if (status <> B_NO_ERROR) then begin
          writeln('Btrieve B_INSERT status = ', status);
        end;
{$endif}
```

```
        end;

        writeln('Inserted ', gneBuffer^.postBuf.numReturned, '
records in new file, status = ', status);
        writeln;
      end;
      fillchar(gneBuffer^, sizeof(GNE_BUFFER), #0);
      gneBuffer^.preBuf.gneHeader.currencyConst := 'EG';
    end;
    dispose(gneBuffer);

    { close open files }
    keyNum := 0;
    if file1Open = TRUE then begin
      dataLen := 0;

      status := BTRVID(
                    B_CLOSE,
                    posBlock1,
                    dataBuffer,
                    dataLen,
                    keyBuf1[1],
                    keyNum,
                    client);

      writeln('Btrieve B_CLOSE status (sample.btr) = ', status);
    end;

    if file2Open = TRUE then begin
      dataLen := 0;
```

```
    status := BTRVID(
                B_CLOSE,
                posBlock2,
                dataBuffer,
                dataLen,
                keyBuf2[1],
                keyNum,
                client);

    writeln('Btrieve B_CLOSE status (sample2.btr) = ', status);
    end;

    { FREE RESOURCES }
    dataLen := 0;
    status := BTRVID(B_STOP, posBlock1, DataBuffer,
                dataLen, keyBuf1[1], 0, client);
    writeln('Btrieve B_STOP status = ', status)

end.
```

Programming Notes

Calling a Btrieve function always returns an INTEGER value that corresponds to a status code. After a Btrieve call, your application should always check the value of this status code. A Status Code of 0 indicates a successful operation. Your application must be able to recognize and resolve a non-zero status.

Although you must provide all parameters on every call, the MicroKernel does not use every parameter for every operation. See Chapter 2, "Btrieve Operations" for a more detailed description of the parameters that are relevant for each operation.

Note

If your application uses Pascal record structures that contain variant strings, consider that odd-length elements in a Pascal record may require an extra byte of storage (even if the record is not packed). This is an important consideration when you define the record length for the Create (14) operation. See your Pascal reference manual for more information on record types.

Visual Basic

This section discusses the following topics:

- ◆ Program Example
- ◆ Compiling, Linking, and Running the Program Example
- ◆ Creating 32-Bit Applications

Program Example

The following example program shows how to perform several of the more common Btrieve operations, and it performs those operations in the order required by the MicroKernel's dependencies (for example, you must open a file before performing I/O to it).

Example A-6   Btrsam32.bas

```
Attribute VB_Name = "BtrSam32"
Rem **************************************************
Rem
Rem Copyright 1998 Pervasive Software Inc. All Rights Reserved
Rem
Rem **************************************************
Rem  BTSAMP32.VBP
Rem  BTRFRM32.FRM
Rem  BTRSAM32.BAS
```

Example A-6   Btrsam32.bas *continued*

```
Rem     This is an example that demonstrates the Visual Basic
Rem     interface to Btrieve.  A VB program must use the
Rem     Windows-specific interface WBTRV32.DLL for 32 bit apps
Rem     and WBTRCALL.DLL for 16 bit applications.
Rem
Rem     You can run this program with VB 4.0 under Windows 3.1,
Rem     Windows NT, or Windows 95.
Rem
Rem     This example creates a Btrieve file with 2 keys and then
Rem      does some insert and read operations on that file.
Rem
Rem     Note about the Btrieve key buffer size: your key buffer
Rem should always be at least as large as the key buffer length
Rem parameter that you supply to Btrieve.  If you indicate that
Rem your key buffer is 10 bytes long and it is only 8, Btrieve
Rem may write past the end of your key buffer when updating
Rem the key buffer with a key value. Note that Btrieve returns
Rem a key value in the key buffer after an 'insert' operation.
Rem
Rem ***********************************************************
Rem     This sample code shows a workaround to the Visual Basic
Rem Long variable type alignment problem. This problem, which
Rem manifests itself as a status 29 on BCREATE operations with
Rem more than one index, as an incorrect value for the total
Rem number of records in a BSTAT operation, and as incorrect
Rem values for data being returned as Long variable types, is
Rem a design limitation of Visual Basic.
Rem
```

```
Rem      We workaround this structure member alignment issue by
Rem using a User Defined Type. This workaround breaks the data
Rem into four (4) units. Each unit is one (1) byte. Once the
Rem data has been returned from the DLL (on BSTAT and in data),
Rem we use byte swapping and conversion from Hexadecimal to
Rem Decimal to return the correct value.
Rem
Rem For more information about the Structure Member Alignment
Rem issue, contact Microsoft or consult the VB4DLL.TXT file
Rem included with Visual Basic.
Rem ********************************************************

DefInt A-Z
Global Const BOPEN = 0
Global Const BCLOSE = 1
Global Const BINSERT = 2
Global Const BUPDATE = 3
Global Const BDELETE = 4
Global Const BGETEQUAL = 5
Global Const BGETNEXT = 6
Global Const BGETGREATEROREQUAL = 9
Global Const BGETFIRST = 12
Global Const BCREATE = 14
Global Const BSTAT = 15
Global Const BSTOP = 25
Global Const BVERSION = 26
Global Const BRESET = 28


Global Const KEY_BUF_LEN = 255

Rem   Key Flags
```

```
Global Const DUP = 1
Global Const MODIFIABLE = 2
Global Const BIN = 4
Global Const NUL = 8
Global Const SEGMENT = 16
Global Const SEQ = 32
Global Const DEC = 64
Global Const SUP = 128

Rem   Key Types
Global Const EXTTYPE = 256
Global Const MANUAL = 512
Global Const BSTRING = 0
Global Const BINTEGER = 1
Global Const BFLOAT = 2
Global Const BDATE = 3
Global Const BTIME = 4
Global Const BDECIMAL = 5
Global Const BNUMERIC = 8
Global Const BZSTRING = 11
Global Const BAUTOINC = 15

Declare Function BTRCALL Lib "wbtrv32.dll" (ByVal OP, ByVal
Pb$, Db As Any, DL As Integer, Kb As Any, ByVal Kl, ByVal Kn)
As Integer
Rem ****************************************************
Rem Structures to overcome problems within Visual Basic.
Rem This User Defined Type allows us to convert the data
Rem coming in from (or going to) our interface.  By treating
Rem the data as a byte we can concatenate the data back
Rem into a Long variable type without conversion problems.
```

```
Type typ_byte4
    fld_Field1   As Byte
    fld_Field2   As Byte
    fld_Field3   As Byte
    fld_Field4   As Byte
End Type
Rem ********************************************************

Rem  Btrieve Structures

Type KeySpec
          KeyPos    As Integer
          KeyLen    As Integer
          KeyFlags  As Integer
          KeyTot    As typ_byte4
          KeyType   As String * 1
          Reserved  As String * 5
End Type

Type FileSpec
          RecLen              As Integer
          PageSize            As Integer
          IndxCnt             As Integer
          NotUsed             As String * 4
          FileFlags           As Integer
          Reserved            As String * 2
          Allocation          As Integer
          KeyBuf0             As KeySpec
          KeyBuf1             As KeySpec
End Type
```

```
Rem Note that due to the way Visual Basic 4.0 handles arrays
Rem of user-defined types, the above type uses
Rem
Rem          KeyBuf0          As KeySpec
Rem          KeyBuf1          As KeySpec
Rem
Rem      rather than
Rem
Rem          KeyBuf(0 To 1)   As KeySpec
Rem
Rem Each Key description must be a separate entry in the
Rem FileSpec.
Rem
Rem      KeyBuf is treated similarly in 'StatFileSpecs', below.
Rem

Type StatFileSpecs
     RecLen              As Integer
     PageSize            As Integer
     IndexTot            As Integer
     RecTot              As typ_byte4
     FileFlags           As Integer
     Reserved            As String * 2
     UnusedPages         As Integer
     KeyBuf0             As KeySpec
     KeyBuf1             As KeySpec
End Type

Type RecordBuffer
         Number          As Double
```

```
          Dummy              As String * 26
End Type

Type VersionBuf
    Major As Integer
    Minor As Integer
    Engine As String * 1
End Type

Global FileBuf As FileSpec
Global DataBuf As RecordBuffer
Global StatFileBuffer As StatFileSpecs
Global PosBlk$
Global BufLen As Integer
Global DBLen As Integer




Sub PrintLB(Item As String)
    BtrFrm32.List1.AddItem Item
End Sub

Sub RunTest()
    PrintLB ("Btrieve Sample Test Started")
    PrintLB ("")

Rem Local variables needed for conversion from byte to long.
Dim loc_RecTot  As Long
Dim h_field1    As String
Dim h_field2    As String
Dim h_field3    As String
```

```
Dim h_field4    As String
Dim h_total     As String

Rem *************************

filename$ = "XFACE.BTR"

PosBlk$ = Space$(128)
KeyBuffer$ = Space$(KEY_BUF_LEN)

Rem
Rem ***************** Btrieve Create ********************
Rem

Rem ************* SET UP FILE SPECS
FileBuf.RecLen = 34
FileBuf.PageSize = 1024
FileBuf.IndxCnt = 2
FileBuf.FileFlags = 0

Rem ************* SET UP KEY SPECS
FileBuf.KeyBuf0.KeyPos = 1
FileBuf.KeyBuf0.KeyLen = 8
FileBuf.KeyBuf0.KeyFlags = EXTTYPE + MODIFIABLE
FileBuf.KeyBuf0.KeyType = Chr$(BFLOAT)

FileBuf.KeyBuf1.KeyPos = 9
FileBuf.KeyBuf1.KeyLen = 26
FileBuf.KeyBuf1.KeyFlags = EXTTYPE + MODIFIABLE + DUP
FileBuf.KeyBuf1.KeyType = Chr$(BSTRING)
```

```
BufLen = Len(FileBuf)
KeyBufLen = Len(filename$)
KeyBuffer$ = filename$
Status = BTRCALL(BCREATE, PosBlk$, FileBuf, BufLen, ByVal
KeyBuffer$, KeyBufLen, 0)

If Status <> 0 Then
    Msg$ = "Error Creating File. Status = " + Str$(Status)
    PrintLB (Msg$)
Else
    Msg$ = "File XFACE.BTR Created Successfully!"
    PrintLB (Msg$)
End If

'Open File
KeyBufLen = KEY_BUF_LEN
KeyBuffer$ = filename$
BufLen = Len(DataBuf)
KeyNum = 0

Status = BTRCALL(BOPEN, PosBlk$, DataBuf, BufLen, ByVal
KeyBuffer$, KeyBufLen, KeyNum)

If Status <> 0 Then
    Msg$ = "Error Opening file! " + Str$(Status)
    PrintLB (Msg$)
    GoTo Fini
Else
    Msg$ = "File Opened Successfully!"
    PrintLB (Msg$)
End If
```

```
'Insert First Record
yr = 1992
mo = 1
dy = 1
DataBuf.Number = DateSerial(yr, mo, dy)
BufLen = Len(DataBuf)
KeyBuffer$ = Space$(KEY_BUF_LEN)
KeyBufLen = KEY_BUF_LEN
DataBuf.Dummy = "first record"

Status = BTRCALL(BINSERT, PosBlk$, DataBuf, BufLen, ByVal
KeyBuffer$, KeyBufLen, 0)

If Status <> 0 Then
    Msg$ = "Error on Insert. " + Str$(Status)
    PrintLB (Msg$)
Else
    Msg$ = "Insert Record #1 Successful!"
    PrintLB (Msg$)
End If

'Insert Second Record
yr = 1993
mo = 1
dy = 1
DataBuf.Number = DateSerial(yr, mo, dy)
BufLen = Len(DataBuf)
KeyBuffer$ = Space$(KEY_BUF_LEN)
KeyBufLen = KEY_BUF_LEN
DataBuf.Dummy = "second record"
```

```
Status = BTRCALL(BINSERT, PosBlk$, DataBuf, BufLen, ByVal
KeyBuffer$, KeyBufLen, 0)

If Status <> 0 Then
    Msg$ = "Error on Insert. " + Str$(Status)
    PrintLB (Msg$)
Else
    Msg$ = "Insert Record #2 Successful!"
    PrintLB (Msg$)
End If

'Insert Third Record
yr = 1994
mo = 1
dy = 1
DataBuf.Number = DateSerial(yr, mo, dy)
BufLen = Len(DataBuf)
KeyBuffer$ = Space$(KEY_BUF_LEN)
KeyBufLen = KEY_BUF_LEN
DataBuf.Dummy = "third record"

Status = BTRCALL(BINSERT, PosBlk$, DataBuf, BufLen, ByVal
KeyBuffer$, KeyBufLen, 0)

If Status <> 0 Then
    Msg$ = "Error on Insert. " + Str$(Status)
    PrintLB (Msg$)
Else
    Msg$ = "Insert Record #3 Successful!"
    PrintLB (Msg$)
```

```
End If

'Get First Record
BufLen = Len(DataBuf)
KeyBuffer$ = Space$(255)
KeyBufLen = KEY_BUF_LEN

Status = BTRCALL(BGETFIRST, PosBlk$, DataBuf, BufLen, ByVal
KeyBuffer$, KeyBufLen, 0)

If Status <> 0 Then
    Msg$ = "Error on BGETFIRST. " + Str$(Status)
    PrintLB (Msg$)
Else
   Msg$ = "BGETFIRST okay for : " + Str$(Year(DataBuf.Number))
+ DataBuf.Dummy
    PrintLB (Msg$)
End If

'Get Next Record
BufLen = Len(DataBuf)
KeyBuffer$ = Space$(KEY_BUF_LEN)
KeyBufLen = KEY_BUF_LEN

Status = BTRCALL(BGETNEXT, PosBlk$, DataBuf, BufLen, ByVal
KeyBuffer$, KeyBufLen, 0)

If Status <> 0 Then
    Msg$ = "Error on BGETNEXT. " + Str$(Status)
    PrintLB (Msg$)
Else
```

```
    Msg$ = "BGETNEXT okay for: " + Str$(Year(DataBuf.Number))
+ DataBuf.Dummy
    PrintLB (Msg$)
End If

'Get Next Record
BufLen = Len(DataBuf)
KeyBuffer$ = Space$(KEY_BUF_LEN)
KeyBufLen = KEY_BUF_LEN

Status = BTRCALL(BGETNEXT, PosBlk$, DataBuf, BufLen, ByVal
KeyBuffer$, KeyBufLen, 0)

If Status <> 0 Then
    Msg$ = "Error on BGETNEXT. " + Str$(Status)
    PrintLB (Msg$)
Else
    Msg$ = "BGETNEXT okay for: " + Str$(Year(DataBuf.Number))
+ DataBuf.Dummy
    PrintLB (Msg$)
End If

'Get Equal
BufLen = Len(DataBuf)
KBuf# = 33604
KeyBufLen = 8

Status = BTRCALL(BGETEQUAL, PosBlk$, DataBuf, BufLen, KBuf#,
KeyBufLen, 0)

If Status <> 0 Then
```

```
     Msg$ = "Error on Get Equal. Status = " + Str$(Status)
     PrintLB (Msg$)
Else
     PrintLB ("BGETEQUAL okay on following key: " +
Str$(DataBuf.Number))
End If

'Stat Call
DBLen = Len(StatFileBuffer)
KeyBuffer$ = Space$(KEY_BUF_LEN)
KeyBufLen = KEY_BUF_LEN

Status = BTRCALL(BSTAT, PosBlk$, StatFileBuffer, DBLen, ByVal
KeyBuffer$, KeyBufLen, 0)

If Status <> 0 Then
  Msg$ = "Error in Stat Call. Status = " + Str$(Status)
  PrintLB (Msg$)
  GoTo Fini
Else
Rem *********************************
Rem Code to work around problems with Visual Basic's "Double
Rem Word" alignment. This code converts the byte data to
Rem hexadecimal, concatenates each byte, and converts it to
Rem decimal for display.
     h_field1 = Hex(StatFileBuffer.RecTot.fld_Field1)
     h_field2 = Hex(StatFileBuffer.RecTot.fld_Field2)
     h_field3 = Hex(StatFileBuffer.RecTot.fld_Field3)
     h_field4 = Hex(StatFileBuffer.RecTot.fld_Field4)
    h_total = "&H" & h_field4 & h_field3 & h_field2 & h_field1
     loc_RecTot = Val(h_total)
```

```
Rem ********************************
    Msg$ = "Number of Records = " & loc_RecTot
    PrintLB (Msg$)
End If

Fini:
Status = BTRCALL(BRESET, PosBlk$, PatientVar, BufLen,
KeyBuffer$, KeyBufLen, KeyNum)

If Status Then
    Msg$ = "Error on B-Reset!" + Str$(Status)
    PrintLB (Msg$)
Else
    Msg$ = "BRESET okay."
    PrintLB (Msg$)
End If

Status = BTRCALL(BSTOP, PosBlk$, PatientVar, BufLen,
KeyBuffer$, KeyBufLen, KeyNum)

If Status Then
    Msg$ = "Error on B-Stop!" + Str$(Status)
    PrintLB (Msg$)
Else
    Msg$ = "BSTOP okay."
    PrintLB (Msg$)
End If
PrintLB ("")
PrintLB ("Btrieve Sample Test Completed")

End Sub
```

Compiling, Linking, and Running the Program Example

➤ **In Visual Basic, to compile, link, and run the program example:**

1. In the Visual Basic programming environment, choose **Open Project** from the **File** menu.

2. Open the appropriate project file in the \Intf\Vb directory:

   ◆ For 32-bit environments, open the BtSamp32.vbp project file.

   ◆ For 16-bit environments, open the BtSamp16.vbp project file.

3. Click the Start button on the toolbar.

   Visual Basic compiles and links the program example and creates a Btrieve Visual Basic Sample window.

4. In the Btrieve Visual Basic Sample window, click the Run Test button.

   Visual Basic runs the program example.

Creating 32-Bit Applications

When developing 32-bit applications with Visual Basic 4.0 and later, a structure alignment issue may cause problems with Btrieve calls. To optimize performance, Visual Basic adds extra bytes into structures to perform double word alignment on some fields in the structure. This is often apparent when a structure contains a Long (4-byte) integer or a single (4-byte) or double (8-byte) precision floating point field. For example:

```
Type DataBuffer
    Fld1      As String * 2
    Fld2      As Long
End Type
```

This structure actually occupies 8 bytes rather than 6, because two extra "dummy" bytes are inserted into the structure to move Fld2 to a double-word boundary. Note that this problem actually exists in most compilers, including C/C++ and Pascal. However, other compilers provide a configuration option you can use to specify the type of structure alignment you want the compiler to use; Visual Basic 4.0 does not provide such a configuration option.

When using a Visual Basic Type structure as a Data Buffer parameter in a Btrieve call, these extra bytes are not apparent to Btrieve. For example, if reading a 6-byte record from a file with a Get First using the structure defined above, Btrieve returns the six bytes of data in the first 6 bytes of the structure; Btrieve does not add any extra bytes, and subsequent access to Fld2 does not produce the correct data.

There are several ways to work around this problem in your Visual Basic application. The sample code in BtrSAM32.BAS is one example of a workaround. Following is an outline of how this workaround uses some intermediate type structures.

1. Declare all type buffers with String fields with the same size as the number field that you would otherwise use.

2. Declare two intermediate type buffers: one that consists of just a 4-byte string field, and one that consists of just a 4-byte long or float field.

3. Use this type buffer from step 1 to read a record from the data file.

4. Copy the 4-byte string from the Btrieve Data Buffer into the intermediate buffer containing only the string field.

5. Use the Visual Basic LSet operation to set the numeric intermediate buffer equal to the string intermediate buffer. (This is equivalent to a C memmove function, in which the bytes are copied regardless of the underlying field types in the structure.)

Similar steps can be used to work around the problem when performing insert or update operations.

appendix B Data Types

The MicroKernel provides standard and extended data types. These types allow the MicroKernel to recognize and collate key values based on the internal storage format of the 18 data types that compilers most commonly use. This capability provides you with greater flexibility in defining keys.

> **Note**
>
> The MicroKernel does not convert data to key types. Each application is responsible for maintaining data typing information.

For historical reasons, the two standard data types, STRING and UNSIGNED BINARY, are also offered as extended data types.

Internally, the MicroKernel compares string keys on a byte-by-byte basis, from left to right. The MicroKernel sorts string keys according to their ASCII value, however, you can define string keys to be case insensitive or to use an ACS.

The MicroKernel compares unsigned binary keys one word at a time. It compares these keys from right to left because the Intel 8086 family of processors reverses the high and low bytes in an integer.

If a particular data type is available in more than one size (for example, both 4- and 8-byte FLOAT values are allowed), the Key Length parameter (used in the creation of a new key) defines the size that will be expected for all values of that particular key. Any attempt to define a key using a Key Length that is not allowed results in a Status 29 (Invalid Key Length).

Table B-1 lists the extended key types and their associated codes.

Table B-1   Extended Key Types and Codes

Type	Code	Type	Code
STRING	0	BFLOAT	9
INTEGER	1	LSTRING	10
FLOAT	2	ZSTRING	11
DATE	3	UNSIGNED BINARY	14
TIME	4	AUTOINCREMENT	15
DECIMAL	5	NUMERICSTS	17
MONEY	6	NUMERICSA	18
LOGICAL	7	CURRENCY	19
NUMERIC	8	TIMESTAMP	20

The following sections, arranged alphabetically by key type, describe the extended key types and their internal storage formats.

AUTOINCREMENT

The AUTOINCREMENT key type is a signed Intel integer that can be either two or four bytes long. Internally, AUTOINCREMENT keys are stored in Intel binary integer format, with the high-order and low-order bytes reversed within a word. The MicroKernel sorts AUTOINCREMENT keys by their absolute values, comparing the values stored in

different records a word at a time from right to left. AUTOINCREMENT keys are incremented each time you insert a record into the file.

The following restrictions apply to AUTOINCREMENT keys:

◆ An AUTOINCREMENT key cannot contain duplicate nonzero key values.

◆ An AUTOINCREMENT key cannot be segmented. However, an AUTOINCREMENT key can be included as a segment of another key, as long as the AUTOINCREMENT key has been defined as a separate, single key first, and the AUTOINCREMENT key number is lower than the segmented key number.

◆ An AUTOINCREMENT key cannot overlap another key.

◆ All AUTOINCREMENT keys must be ascending.

The MicroKernel treats AUTOINCREMENT key values as follows when you insert records into a file:

◆ If you specify a value of binary 0 for the AUTOINCREMENT key, the MicroKernel assigns a value to the key based on the following criteria:

  ◆ If you are inserting the first record in the file, the MicroKernel assigns the value of 1 to the AUTOINCREMENT key.

  ◆ If records already exist in the file, the MicroKernel assigns the key a value that is one number higher than the highest existing absolute value in the file.

◆ If you specify a nonzero value for the AUTOINCREMENT key, the MicroKernel inserts the record into the file and uses the specified value as the key value. If a record containing that value already exists in the file, the MicroKernel returns an error status code and does not insert the record.

When you delete a record containing an AUTOINCREMENT key, the MicroKernel completely removes the record from the file. The MicroKernel does not reuse the deleted key value unless you specify that value when you insert another record into the file.

As mentioned previously, the MicroKernel always sorts AUTOINCREMENT keys by their absolute values. For example, you can do the following:

- Specify a negative value for an AUTOINCREMENT key when you insert a record.
- Update a record and negate the value for the AUTOINCREMENT key.

In any case, the MicroKernel sorts the key according to its absolute value. This allows you to use negation to flag records without altering the record's position in the index. In addition, when you perform a Get operation and specify a negative value in the key buffer, the MicroKernel treats the negative value as the absolute value of the key.

You can initialize the value of a field in all or some records to zero and later add an index of type AUTOINCREMENT. This feature allows you to prepare for an AUTOINCREMENT key without actually building the index until it is needed.

When you add the index, the MicroKernel changes the zero values in each field appropriately, beginning its numbering with a value equal to the greatest value currently defined in the field, plus one. If nonzero values exist in the field, the MicroKernel does not alter them. However, the MicroKernel returns an error status code if nonzero duplicate values exist in the field.

BFLOAT

The BFLOAT key type is a single or double-precision real number. A single-precision real number is stored with a 23-bit mantissa, an 8-bit exponent biased by 128, and a sign bit. The internal layout for a 4-byte float is as follows:

8-bit exponent   23-bit mantissa
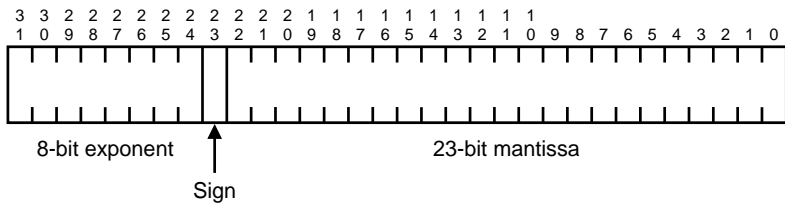
Sign

The representation of a double-precision real number is the same as that for a single-precision real number, except that the mantissa is 55 bits instead of 23 bits. The least significant 32 bits are stored in bytes 0 through 3. The BFLOAT type is commonly used in BASIC applications.

CURRENCY

The CURRENCY data type represents an 8-byte signed quantity, sorted and stored in Intel binary integer format; therefore, its internal representation is the same as an 8 byte INTEGER data type. The CURRENCY data type has an implied four digit scale of decimal places, which represents the fractional component of the currency data value.

DATE

The DATE key type is stored internally as a 4-byte value. The day and the month are each stored in 1-byte binary format. The year is a 2-byte binary number that represents the

entire year value. The MicroKernel places the day into the first byte, the month into the second byte, and the year into a two-byte word following the month.
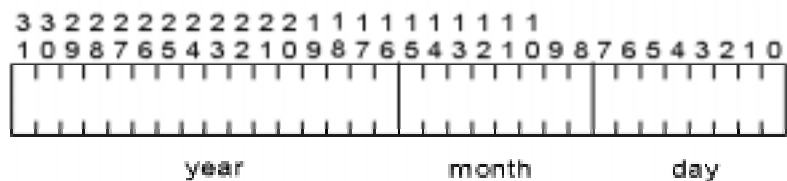
DECIMAL

The DECIMAL key type is stored internally as a packed decimal number with two decimal digits per byte. The internal representation for an n-byte DECIMAL field is as follows:
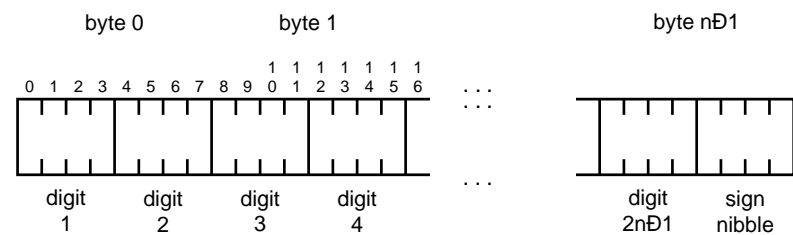
The sign nibble is either $0xF$ or $0xC$ for positive numbers and $0xD$ for negative numbers. The decimal point is implied; no decimal point is stored in the DECIMAL field. Your application is responsible for tracking the location of the decimal point for the value in a DECIMAL field. All the values for a DECIMAL key type must have the same number of decimal places in order for the MicroKernel to collate the key correctly. The DECIMAL type is commonly used in COBOL applications.

FLOAT

The FLOAT key type is consistent with the IEEE standard for single and double-precision real numbers. The internal format for a 4-byte FLOAT consists of a 23-bit mantissa, an 8-bit exponent biased by 127, and a sign bit, as follows:

```
3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
```

8-bit exponent   23-bit mantissa
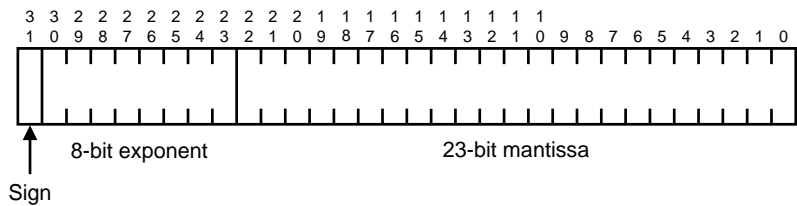
Sign

A FLOAT key with 8 bytes has a 52-bit mantissa, an 11-bit exponent biased by 1023, and a sign bit. The internal format is as follows:

bytes 7-4:

```
6 6 6 6 5 5 5 5 5 5 5 5 5 5 4 4 4 4 4 4 4 4 3 3 3 3 3 3 3 3
3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2
```

11-bit exponent   20-bit mantissa

Sign

bytes 3-0:

```
3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
```

32-bit mantissa
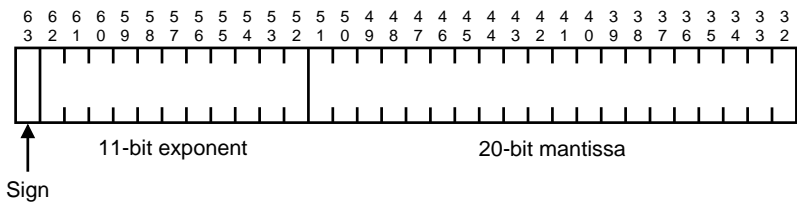
INTEGER

The INTEGER key type is a signed whole number and must contain an even number of bytes. Internally, INTEGER fields are stored in Intel binary integer format, with the high-order and low-order bytes reversed within a word. The MicroKernel evaluates the key from right to left, a word at a time. The sign must be stored in the high bit of the rightmost byte. The INTEGER type is commonly used in C applications.

Length in Bytes	Value Ranges
1	0 – 255
2	-32768 – 32767
4	-2147483648 – 2147483647
8	-9223372036854775808 – 9223372036854775807

LOGICAL

The LOGICAL key type is stored as a 1 or 2-byte value. The MicroKernel collates LOGICAL key types as strings. Doing so allows your application to determine the stored values that represent true or false.

LSTRING

The LSTRING key type has the same characteristics as a regular STRING type, except that the first byte of the string contains the binary representation of the string's length. The length stored in byte 0 of an LSTRING key determines the number of significant bytes.

The MicroKernel ignores any values beyond the specified length of the string. The LSTRING type is commonly used in Pascal applications.

MONEY

The MONEY key type has the same internal representation as the DECIMAL type.

NUMERIC

NUMERIC values are stored as ASCII strings, right justified with leading zeros. Each digit occupies one byte internally. The rightmost byte of the number includes an embedded sign with an EBCDIC value. Table B-2 indicates how the rightmost digit is represented when it contains an embedded sign for positive and negative numbers.

Table B-2   Rightmost Digit with Embedded Sign

Digit	Positive	Negative
1	A	J
2	B	K
3	C	L
4	D	M
5	E	N
6	F	O
7	G	P
8	H	Q
9	I	R
0	{	}

For positive numbers, the rightmost digit can be represented by 1 through 0 instead of A through {. The MicroKernel processes positive numbers represented either way. The NUMERIC type is commonly used in COBOL applications.

NUMERICSA

The NUMERICSA key type (sometimes called NUMERIC SIGNED ASCII) is a COBOL data type that is the same as the NUMERIC data type, except that the embedded sign has an ASCII value instead of an EBCDIC value.

NUMERICSTS

The NUMERIC STS key type (sometimes called SIGN TRAILING SEPARATE) is a COBOL data type that has values resembling those of the NUMERIC data type. NUMERICSTS values are stored as ASCII strings and right justified with leading zeros. However, the rightmost byte of a NUMERICSTS string is either "+" (ASCII 0x2B) or "-" (ASCII 0x2D). This differs from NUMERIC values that embed the sign in the rightmost byte along with the value of that byte.

STRING

The STRING key type is a sequence of characters ordered from left to right. Each character is represented in ASCII format in a single byte, except when the MicroKernel is determining whether a key value is null.

TIME

The TIME key type is stored internally as a 4-byte value. Hundredths of a second, second, minute, and hour values are each stored in 1-byte binary format. The MicroKernel places the hundredths of a second value into the first byte, followed respectively by the second, minute, and hour values.

TIMESTAMP

The TIMESTAMP data type represents a time and date value. In applications that use Pervasive's Scalable SQL or ODBC Interface, use this data type to stamp a record with the time and date of the last update to the record. TIMESTAMP values are stored in 8-byte unsigned values representing septa seconds (10^{-7} second) since January 1, 0001 in a Gregorian calendar.

TIMESTAMP is intended to cover time and data values made up of the following components: year, month, day, hour, minute, and second. The following table indicates the valid values of each of these components:

YEAR	0001 to 9999
MONTH	01 to 12
DAY	01 to 31, constrained by the value of MONTH and YEAR in the Gregorian calendar.

HOUR	00 to 23
MINUTE	00 to 59
SECOND	00 to 59

The Programming Interfaces installation option contains sample conversion functions that can be integrated into applications which access Btrieve data using Pervasive's Scalable SQL or ODBC Interface, or vice versa. These functions include: 1) converting the number of septa seconds since January 1, 0001 to a date and time; and 2) converting a date and time to the number of septa seconds since January 01, 0001.

Refer to the Programming Interfaces on-line samples for these two functions. Also refer to the *SQL Language Reference* for additional documentation on using the TIMESTAMP data type in SQL environments.

UNSIGNED BINARY

The MicroKernel sorts UNSIGNED BINARY keys as unsigned INTEGER keys. An UNSIGNED BINARY key must contain an even number of bytes (2, 4, 6, and so on). The MicroKernel compares UNSIGNED BINARY keys a word at a time, from right to left.

An UNSIGNED BINARY key is sorted in the same manner as an INTEGER key. The differences between an UNSIGNED BINARY key and an INTEGER key are that an INTEGER has a sign bit, while an UNSIGNED BINARY type does not, and an UNSIGNED BINARY key can be longer than 4 bytes.

ZSTRING

The ZSTRING key type corresponds to a C string. It has the same characteristics as a regular string type except that a ZSTRING type is terminated by a byte containing a

binary 0. The MicroKernel ignores any values beyond the first binary 0 it encounters in the ZSTRING, except when the MicroKernel is determining whether a key value is null.

appendix # C Quick Reference of Btrieve Operations

This appendix provides a summary of Btrieve operations in numerical order by operation code.

Table C-1   Quick Reference of Btrieve Operations

Operation	Code	Description
Open	0	Makes a file available for access.
Close	1	Releases a file from availability.
Insert	2	Inserts a new record into a file.
Update	3	Updates the current record.
Delete	4	Removes the current record from the file.
Get Equal	5	Returns the record whose key value matches the specified key value.
Get Next	6	Returns the record following the current record in the index path.
Get Previous	7	Returns the record preceding the current record in the index path.
Get Greater Than	8	Returns the record whose key value is greater than the specified key value.
Get Greater Than or Equal	9	Returns the record whose key value is equal to or greater than the specified key value.

Table C-1   Quick Reference of Btrieve Operations *continued*

Operation	Code	Description
Get Less Than	10	Returns the record whose key value is less than the specified key value.
Get Less Than or Equal	11	Returns the record whose key value is equal to or less than the specified key value.
Get First	12	Returns the first record in the specified index path.
Get Last	13	Returns the last record in the specified index path.
Create	14	Creates a file with the specified characteristics.
Stat	15	Returns file and index characteristics, and number of records.
Extend	16	Divides a data file over two logical disk drives. *This operation is not supported in Btrieve 6.0 and later.*
Set Directory	17	Changes the current directory.
Get Directory	18	Returns the current directory.
Begin Transaction	19 1019	Marks the beginning of a set of logically related operations. Operation 19 begins an exclusive transaction. Operation 1019 begins a concurrent transaction.
End Transaction	20	Marks the end of a set of logically related operations.
Abort Transaction	21	Removes operations performed during an incomplete transaction.

Table C-1   Quick Reference of Btrieve Operations *continued*

Operation	Code	Description
Get Position	22	Returns the position of the current record.
Get Direct/Chunk	23	Returns data from the specified portions (chunks) of a record at a specified position.
Get Direct/Record	23	Returns the record at a specified position.
Step Next	24	Returns the record from the physical location following the current record.
Stop	25	Terminates the workstation MicroKernel (not available for server-based MicroKernels).
Version	26	Returns the version number of the MicroKernel.
Unlock	27	Unlocks a record or records.
Reset	28	Releases all resources held by a client.
Set Owner	29	Assigns an owner name to a file.
Clear Owner	30	Removes an owner name from a file.
Create Index	31	Creates an index.
Drop Index	32	Removes an index.
Step First	33	Returns the record in the first physical location in the file.
Step Last	34	Returns the record in the last physical location in the file.

Table C-1   Quick Reference of Btrieve Operations *continued*

Operation	Code	Description
Step Previous	35	Returns the record in the physical location preceding the current record.
Get Next Extended	36	Returns one or more records that follow the current record in the index path. Filtering conditions can be applied.
Get Previous Extended	37	Returns one or more records that precede the current record in the index path. Filtering conditions can be applied.
Step Next Extended	38	Returns one or more successive records from the location physically following the current record. Filtering conditions can be applied.
Step Previous Extended	39	Returns one or more preceding records from the location physically preceding the current record. Filtering conditions can be applied.
Insert Extended	40	Inserts one or more records into a file.
Continuous Operation	42	Server-based MicroKernels only. Places the specified file in or removes the file from continuous operation mode, for use in system backups.
Get By Percentage	44	Returns the record located approximately at a position derived from the specified percentage value.
Find Percentage	45	Returns a percentage figure based on the current record's position in the file.
Get Key	+50	Detects the presence of a key value in a file, without returning an actual record.

Table C-1   Quick Reference of Btrieve Operations *continued*

Operation	Code	Description
Update Chunk	53	Updates specified portions (chunks) of the current record. This operation can also append data to a record or truncate a record.
Stat Extended	65	Returns filenames and paths of an extended file's components and reports whether a file is using a system-defined log key.
Single-record wait lock	+100	Locks only one record at a time. If the record is already locked, the MicroKernel retries the operation.
Single-record no-wait lock	+200	Locks only one record at a time. If the record is already locked, the MicroKernel returns an error status code.
Multiple-record wait lock	+300	Locks several records concurrently in the same file. If the record is already locked, the MicroKernel retries the operation.
Multiple-record no-wait lock	+400	Locks several records concurrently in the same file. If the record is already locked, the MicroKernel returns an error status code.
No-wait page lock	+500	In a concurrent transaction, tells the MicroKernel not to wait if the page to be changed has already been changed by another active concurrent transaction. This bias can be combined with any of the record locking biases (+100, +200, +300, or +400).

Index

A

Abort Transaction operation 36
Accelerated file open mode 154
ACS. *See* Alternate collating
    sequence.
Adding an index to an existing file 67
ALT constant 58
Alternate collating sequence
    Create operation and 59
    creating keys that use 58
    flag in Stat operation for 173
    ISR 64
    locale-specific 63
    user-defined 62
AND and OR operations in filters 129
Ascending sort order 58
Attributes
    file 54
    key 58
AUTOINCREMENT data type 338

B

Balanced indexes 54
BALANCED_KEYS constant 54
Begin Transaction operation 38
BFLOAT data type 341
BIN constant 58
Blank truncation 54
BLANK_TRUNC constant 54
BRQSHELLINIT function 15
BTRCALL function 14
BTRCALL32 function 15
BTRCALLBACK function 15
BTRCALLID function 15
BTRCALLID32 function 15
Btrieve
    function parameters 16
    functions 13
    operations, using 33
BTRSAMP.C 226
BTRV function 14
BTRVID function 14

L

Linked duplicate keys 58
Linking a 32-bit extended DOS
      application with the C interface
      246
Locale-specific ACSs 63
Lock biases 30
LOGICAL data type 344
LSTRING data type 344

M

Manipulation operations 26
MANUAL_KEY constant 58
MEFS bias with file open modes 155
MOD constant 58
Modifiable keys 58
Modification operations 26
MONEY data type 345
Multiple record locks 196

N

NAMED_ACS constant 58
Next-in-record subfunction bias 98,
      210

NO_INCLUDE_SYSTEM_DATA
      constant 54
NOCASE_KEY constant 58
No-currency-change
   Insert Extended operation and 147
   Insert operation and 145
   Update operation and 197
Normal file open mode 153
NUL constant 58
Null keys 58
NUMBERED_ACS constant 58
NUMERIC data type 345
NUMERICSA data type 346
NUMERICSTS data type 346

O

Obsolete functions 15
Open
   modes 153
   operation 152
Operation Code parameter 17
Operations, using in applications 33
Owner names
   clearing 41
   setting 165

P

Page preallocation 54
Pascal language interface 297
Pointers, reserved duplicate 54
Position Block parameter 19
PRE_ALLOC constant 54

R

Random chunk descriptors 91, 203
Read-only file open mode 154
Rebuilding a damaged index 76
Records
  deleting 73
  inserting 144
  inserting multiple 147
  updating 197
Rectangle chunk descriptors 95, 206
REPEAT_DUPS_KEY constant 58
Repeating duplicate keys 58
Requester version, retrieving 213
Reserved duplicate pointers 54
Reset operation 161
Resources, releasing 161
Retrieval operations 26
Retrieving records

equal to the index path 105
first in physical location 179
first in the index path 107
Get By Percentage operation and
    84
Get Direct/Record operation and
    101
getting one or more chunks of a
    record 89
greater than or equal to the index
    path 111
greater than the index path 109
last in physical location 181
last in the index path 116
less than or equal to the index path
    120
less than the index path 118
next in physical location 183, 185
next in the index path 122, 125
previous in physical location 189,
    191
previous in the index path 138, 141
using established physical location
    136
RQSHELLINIT function 15

S

T

U

V

W

Z

User Comments

Pervasive Software would like to hear your comments and suggestions about our manuals. Please write your comments below and send them to us at:

Pervasive Software Inc.
Documentation
8834 Capital of Texas Highway
Austin, Texas 78759 USA

Pervasive.SQL 7
Btrieve Programmer's Reference
Part # 100-003411-002
February 1998

Telephone: 1-800-287-4383
Fax: 512-794-1778
Email: docs@pervasive.com

Your name and title: _____
Company: _____
Address: _____

_____

Phone number: _____ Fax: _____

You may reproduce these comment pages as needed so that others can send in comments also.

I use this manual as: ❏ an overview ❏ a tutorial ❏ a reference ❏ a guide

	Excellent	Good	Fair	Poor
Completeness	❏	❏	❏	❏
Readability (style)	❏	❏	❏	❏
Organization/Format	❏	❏	❏	❏
Accuracy	❏	❏	❏	❏
Examples	❏	❏	❏	❏
Illustrations	❏	❏	❏	❏
Usefulness	❏	❏	❏	❏

Please explain any of your above ratings: _____
_____
_____
_____

In what ways can this manual be improved?_____
_____
_____
_____
_____
_____

You may reproduce these comment pages as needed so that others can send in comments also.