ACTIAN™

Btrieve API Guide

Zen v15

Activate Your Data™

Contents

About This Document

This documentation is a guide to the Btrieve application programming interface (API).

Who Should Read This Document

This document is designed for any user who is familiar with Zen and wants to develop applications that use the Btrieve API.

Actian Corporation would appreciate your comments and suggestions about this manual. As a user of our documentation, you are in a unique position to provide ideas that can have a direct impact on future releases of this and other manuals. If you have comments or suggestions for the product documentation, post your request at the Community Forum on the Zen website.

Introduction to Btrieve APIs

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

The Btrieve API provides compatibility with the following programming languages and development environments:

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

The following topics cover the API functionality:

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

You can also go directly to a list of Btrieve API Operations or the Quick Reference of Btrieve Operations.

Btrieve API Functions

The Btrieve API is *single-function* in that 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 API function.

The following table lists Btrieve API functions provided for use with the operation codes.

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. Uses older data buffer layout.
BTRCALL BTRCALLID	All	Use when you want to specify the key length argument. Uses older data buffer layout.
BTRVEX BTRVEXID	All	Use when you need longer data buffers, or when you want the new data buffer layout. May be intermixed with BTRV type entry points so long as you interpret the data buffers correctly.

To find the language-specific syntax for calling Btrieve API functions, see Btrieve API Programming in *Zen Programmer's Guide*.

BTRV

BTRV allows an application to make calls to the MicroKernel Engine. All the language interface modules provided with the Programming Interfaces installation option support the BTRV function. In some cases, the BTRV function actually calls the BTRCALL function. However, BTRV is the preferred function because of the platform independence it provides.

BTRVID

BTRVID allows an application to make a single MicroKernel Engine 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 the MicroKernel Engine and to execute operations for one client without affecting the state of the other clients. For more information, see Client ID.

BTRCALL

For Windows, Linux, and macOS, BTRCALL is equivalent to the BTRV function. You should use the BTRV function instead of BTRCALL unless you cannot afford the slight performance decrease that occurs with BTRV.

BTRCALLID

Use the BTRCALLID function if you need client-level control and your application operates in Windows, Linux, macOS, or Raspbian.

This function is similar to the BTRVID function, except that it does not call an intermediate function.

Note: The legacy BTRCALLID32 function is aliased to the BTRCALLID function.

BTRVEX

The potentially large size of 13.0 format files and the use of larger data buffers requires run-time values larger than what older Btrieve interfaces have provided. The newer entry points BTRVEX and BTRVEXID meet these requirements. They are similar to BTRCALL and BTRCALLID, except that some of the function arguments use wider types, and some of the data buffers are laid out differently. The declarations are in btrvexid.h and the implementations are in the same files as BTRCALL.

For a Btrieve operation that uses a BTRVEX entry point, some values passed in data buffers are wider, such as 8-byte record addresses and record counts. Please note that the 8-byte behavior is due to the BTRVEX entry points and does not depend on the format version of the file being accessed. To use BTRVEX, the following operations require changed data buffer layouts:

- Create (14), Create Index (31)
- Stat (15)
- Get Position (22)
- Get Direct (23)
- Get Next Extended (36), Get Previous Extended (37)
- Step Next Extended (38), Step Previous Extended (39)
- Insert Extended (40)
- Find Percentage (45)
- Stat Extended (65) subfunctions 3 and 8
- Unlock (27)

The choice of entry point does not affect record data.

As noted above, the data buffer size argument for BTRVEX is a pointer to a 32-bit integer, where BTRCALL uses a 16-bit integer. Thus, data buffers can be larger than 64 KB.

If you are migrating to the new file format, keep in mind that the position block and client ID values can be used with both BTRCALL and BTRVEX, so it is not necessary to convert all code to BTRVEX at once.

The key number argument for BTRVEX is a 32-bit integer, where BTRCALL uses an 8-bit signed integer. To make it easier to convert existing code to BTRVEX, the BTRVEX entry point remaps key values 128 through 255 to –128 through –1. This accommodates constants that were specified as unsigned bytes (e.g., 0xFE) instead of as signed bytes (e.g., –2).

BTRVEXID

Like BTRVID and BTRCALLID, BTRVEXID allows client ID control in addition to the benefits of BTRVEX.

Obsolete Functions

The following historical functions are supported to maintain compatibility with applications written for previous Btrieve API 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 MicroKernel versions.

Btrieve API Function Parameters

Every function call must provide all parameters. This holds true even though the MicroKernel Engine does not use every parameter on every operation and in some cases may ignore a parameter value. In general, different parameters can be sent and returned for each operation. See Btrieve API Operations for details of the parameters for each Btrieve API operation.

Note: *C developers:* See the file btitypes.h for information about the platform-independent data types and pointers used in the C language interface.

Btrieve API functions use the following parameters:

- Operation Code
- Status Code (BASIC and COBOL only)
- Position Block
- Data Buffer
- Data Buffer Length
- Key Buffer
- Key Number
- Client ID (BTRVID, BTRCALLID, and BTRVEXID functions only)
- Key Length (BTRCALL, BTRCALLID, BTRVEX, and BTRVEXID functions only)

Operation Code

The operation code parameter determines the action of the Btrieve API function. For example, the operation may read, write, delete, or update one or more records. Your application must specify an operation code in every Btrieve API call. The MicroKernel Engine never changes the code. Operation codes are described under Btrieve API Operations.

Note: *C developers:* The data type of the variable you specify must be either BTI_WORD, an unsigned short integer, or BTI_INT, a signed 32-bit integer used only with BTRVEX and BTRVEXID. In both cases, the variable is passed by value.

Status Code

In BASIC and COBOL applications, the MicroKernel Engine returns status codes, which are signed integers. In most programming environments, the status code is the return value of the Btrieve API function call. However, some BASIC and COBOL language interfaces require a Status Code parameter, which contains a coded value to indicate whether errors occurred during the operation. After a Btrieve API call, the application must always check the value of the status variable to determine success.

Zen 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

Here 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 ensure your application's future stability.

Note: In the older BTRV functions, status codes are 2-byte integers, while the newer BTRVEX and BTRVEXID return 4-byte integers.

Position Block

The position block parameter is the address of a 128-byte array that the MicroKernel Engine 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. The MicroKernel Engine 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 API 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 Engine uses the position block to determine which file a particular call is for. Similarly, when you open the same file more than once, the engine uses a different position block for each Open operation. Likewise, the engine uses a different position block for each separate client that opens the same file. Clients cannot share position blocks.

Note: The position block is not bound to an entry point. It is possible to open a data file using BTRV, read data using BTRVEX, and close the file using BTRCALL.

Data Buffer

Your application transfers data to and from a file using the data buffer. The information passed to or from the MicroKernel Engine in the data buffer depends on which Btrieve API 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 API operation, the data buffer can contain other information, such as file or key specifications, MicroKernel Engine 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 larger than the allocated size of your data buffer, MicroKernel Engine modification operations may destroy data following the data buffer.

Note: The same operations use different layouts depending on the entry point. BTRV, BTRVID, BTRCALL, and BTRCALLID use the legacy layout. BTRVEX and BTRVEXID use a newer, slightly different layout. The different layouts do not affect user data records.

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:* Applications must pass the data buffer length parameter ByRef as a Long integer.
C, COBOL, and Pascal developers: Applications must pass the data buffer length parameter as a pointer to a 2-byte integer for older BTRV functions, while the newer BTRVEX and BTRVEXID functions use a 4-byte 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 the maximum data buffer size, you must use a chunk operation to operate on a portion of the record.

The MicroKernel Engine 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 Engine to overwrite memory. The data buffer length should always represent the size of the allocated data buffer.

Note: The data buffer length is 2 bytes for the older BTRV functions and 4 bytes for the newer BTRVEX and BTRVEXID. For the older functions, the maximum data buffer size is 64 KB, while for the two newer ones it is 252 KB.

Key Buffer

Your application must pass the key buffer parameter on every Btrieve API 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 API 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 API 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 Engine returns an error if the string variable passed as the key buffer is shorter than the defined key length. If your first application call does not require initialization of the key buffer, then assign the string variable the value SPACE$(*x*), where *x* represents the defined length of the key. Until your application assigns a value in BASIC to the string variable, its length is 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 Engine cannot determine the key buffer length when an application makes a Btrieve API call. You must ensure that the buffer is at least as long as the key length you chose when you created the key. Otherwise, Btrieve operations may destroy data stored in memory following the key buffer. It is best to have a 255-byte key buffer, because 255 is the maximum key length.

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. In all functions, the key number has a value range of 0 through 118.

The key number size varies:

- For BTRV and BTRVID, this parameter is a 2-byte integer.
- For BTRCALL, BTRCALLID, BTRCALL32, and BTRCALLID32, it is a 1-byte signed character (BTI_CHAR).
- For BTRVEX and BTRVEXID, it is a 4-byte integer. As a convenience for code migrating to BTRVEX, the key values 128–255 are mapped to key values –128 to –1. This simulates the conversion of a large unsigned byte value (e.g., 0xFF) to the signed byte argument of BTRCALL.

Btrieve API functions never alter the key number parameter.

Other information can be sent or returned in the key number parameter, such as a value indicating the mode for opening the file.

Client ID

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

Element	Length (bytes)	Description	
Filler	12	Initialize to 0.	
Service Agent ID	2	Identifies each instance of your application to the MicroKernel Engine. 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 Engine assumes special meaning for the following values:	
		0x4140 (@A)	Used internally.
		0xFFFF	Used internally.
		0x4952 (RI)	Used internally.

Element	Length (bytes)	Description	
		0x5244 (DR)	Used internally.
		0x4553 (SE) 0x4353 (SC) 0x4344 (DC) 0x4544 (DE) 0x5544 (DU)	Used to identify clients originated by Scalable SQL.
		0x5257 (WR)	Used by Btrieve Requesters.
Client Identifier	2	Establishes a client identity within the current instance of your application. The MicroKernel Engine uses this unique identifier for concurrency and transaction-processing purposes.	

Key Length

The Key Length parameter is used only for BTRCALL, BTRCALLID, BTRVEX, and BTRVEXID.

The key length value is used as follows:

- For BTRCALL and BTRCALLID, pass the key length as an unsigned char of type BTI_BYTE, with a value of the allocated length of your key buffer. The maximum length you can specify is 255, the maximum length of any key.
- For BTRVEX and BTRVEXID, pass the key length as an unsigned char of type BTI_INT, with the same maximum length of 255.

Consider the following when you access the key buffer:

- For all four functions, the bytes of the key buffer up to the specified key length must be readable or writable, depending on the operation code.
- BTRV and BTRVID assume a key length of 255, so you should supply a key buffer at least that large.
- For BTRCALL and BTRCALLID, Zen client components may attempt to determine the actual key length and in some instances may read or write a smaller portion of the key buffer.
- BTRVEX and BTRVEXID take the specified key length at face value.

Summary of Btrieve API Operations

The Btrieve API provides over 40 operations to call from your application program. This topic summarize these operations. See Btrieve API Operations for complete descriptions. See Quick Reference of Btrieve Operations for brief summaries ordered by operation code.

Session-Specific Operations

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

Operation	Code	Description
Stop	25	Terminates the workstation MicroKernel Engine (not available for server-based MicroKernel Engine).
Version	26	Returns the version number of the MicroKernel Engine.
Reset	28	Releases all resources held by a client.
Set Directory	17	Sets the current directory to a specified path name.
Get Directory	18	Returns the current directory for a specified logical disk drive.
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.
Continuous Operation	42	Allows you to perform system backups without closing active MicroKernel Engine files.

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
- Data Retrieval
- Data Manipulation

File Access and Information

The following table lists access and information operations to create a file, open and close it, retrieve its statistics, set and clear its owner name, start or stop its continuous operation mode, unlock it, and create and drop its indexes.

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.
Continuous Operation	42	Allows you to perform system backups without closing active MicroKernel Engine files.
Stat Extended	65	Returns file names and paths of components of an extended file 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.
Unlock	27	Unlocks a record or records.
Create Index	31	Creates an index.
Drop Index	32	Removes an index.

Data Retrieval

The following table lists data retrieval operations to retrieve a single record or a set of records given specified criteria. The Btrieve API supports data retrieval either by logical location in an index path or by physical location. For more about accessing records, see *Zen Programmer's Guide*.

In addition, you can apply biases to the operation codes to control file and record locking in multiclient situations. For more about supporting multiple clients, see *Zen Programmer's Guide*.

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.

Operation	Code	Description
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.
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 position of the current record 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.

Operation	Code	Description
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.
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 position of the current record in the file.
Concurrency Control Biases (Add to the Appropriate Operation Code)		
Single-record wait read lock	+100	Locks only one record at a time. If the record is already locked, the client retries the operation.
Single-record no-wait read lock	+200	Locks only one record at a time. If the record is already locked, the MicroKernel Engine returns an error status code.
Multiple-record wait read lock	+300	Locks several records concurrently in the same file. If the record is already locked, the client retries the operation.
Multiple-record no-wait read lock	+400	Locks several records concurrently in the same file. If the record is already locked, the MicroKernel Engine returns an error status code.
No-wait page write lock	+500	In a concurrent transaction, tells the MicroKernel Engine 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 read biases (+100, +200, +300, or +400).

Data Manipulation

The following table lists operations to insert, update, or delete data.

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.
Get Next Delete Extended	85	Removes records matching a filter condition, from the logical next position to the end of the file.
Get Previous Delete Extended	86	Removes records matching a filter condition, from the logical previous position to the beginning of the file.
Step Next Delete Extended	87	Removes records matching a filter condition, from the physical next position to the end of the file.
Step Previous Delete Extended	88	Removes records matching a filter condition, Removes records matching a filter condition, from the physical previous position to the beginning of the file.

Unsupported Operations

When looking at MicroKernel Engine traces or SDK header files, you may see operations that are not listed in the reference for Btrieve API operations. These are for the internal use of Zen and you should not use them in your applications. The following operations are not supported.

Operation	Code	Description
B_MISC_DATA	41	Reserved for use by MicroKernel Engine
B_EXTEND	16	Reserved for use by SQL engine
Begin Transaction (nested) via Btrieve	2019	Reserved for use by MicroKernel Engine

Sequence of Events in Performing a Btrieve API Operation

To perform a Btrieve API 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 (0) operation on that file.
2. Initialize the parameters that the Btrieve API operation requires. The parameters are program variables or data structures that correspond in type and size to the particular values that the MicroKernel Engine 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 API function. (Refer to Btrieve API Functions.)
4. Evaluate the results of the function call. Every Btrieve API 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 API Operations

The operations documented here are the ones your application can perform using the Btrieve API. For each operation you will find 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 succeeds – for example, status code 60 means the specified reject count has been reached.
- Positioning – The effect the operation has on the logical or physical currency of the records in a file.

The Btrieve API operations are listed alphabetically.

- Find Percentage (45)
- Get By Percentage (44)
- Get Direct/Chunk (23)
- Get Direct/Record (23)
- Get Directory (18)
- Get Equal (5)
- Get First (12)
- Get Greater Than (8)
- Get Greater Than or Equal (9)
- Get Key (+50)
- Get Last (13)
- Get Less Than (10)
- Get Less Than or Equal (11)
- Get Next (6)
- Get Next Delete Extended (85)
- Get Next Extended (36)
- Get Position (22)
- Get Previous (7)
- Get Previous Delete Extended (86)
- Get Previous Extended (37)
- Insert (2)
- Insert Extended (40)
- Login/Logout (78)
- Open (0)
- Reset (28)
- Set Directory (17)
- Set Owner (29)
- Stat (15)
- Stat Extended (65)

- Step First (33)
- Step Last (34)
- Step Next (24)
- Step Next Delete Extended (87)
- Step Next Extended (38)
- Step Previous (35)
- Step Previous Delete Extended (88)
- Step Previous Extended (39)
- Stop (25)
- Unlock (27)
- Update (3)
- Update Chunk (53)
- Version (26)

Abort Transaction (21)

The Abort Transaction operation (B_ABORT_TRAN) 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

	Op Code	Pos Block	Data Buf	Data Buf Len	Key Buffer	Key Number
Sent	**X**					
Returned						

Prerequisites

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

Procedure

Set the operation code to 21. The MicroKernel Engine ignores all other parameters on an Abort Transaction call.

Result

If the Abort Transaction operation succeeds, the MicroKernel Engine 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 fails, the MicroKernel Engine 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 (B_BEGIN_TRAN) defines the start of a transaction. Transactions are useful when you need to perform multiple Btrieve API 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 Engine does not permanently complete any operations in the set unless you request the completion with an explicit End Transaction (20). Changes made within a transaction are not visible to other users until the End Transaction operation succeedsly performed.

The MicroKernel Engine prohibits certain operations during transactions because they have too great an effect on the file or on performance. These operations include Set Owner (29), Clear Owner (30), Create Index (31), and Drop Index (32).

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	Key Buffer	Key Number
Sent	**X**					
Returned						

Prerequisites

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

Procedure

Set the operation code to 19 to begin an exclusive transaction, or 1019 to begin a concurrent transaction. The MicroKernel Engine 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 Engine 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. 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.

Note: For details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations Guide*.

Result

If the Begin Transaction operation succeeds, the MicroKernel Engine returns status code 0.

If the Begin Transaction operation fails, the MicroKernel Engine 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 (B_CLEAR_OWNER) 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 Engine decrypts the file during a Clear Owner operation. For more information, see Owner Names in *Advanced Operations Guide*.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	Key Buffer	Key Number
Sent	**X**	**X**				
Returned		**X**				

Prerequisites

- The file must be open, and an owner name must have been specified.
- 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 Engine 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 Engine 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 fails, the MicroKernel Engine returns one of the following status codes:

3	The file is not open.
41	The MicroKernel Engine does not allow the attempted operation.

Positioning

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

Close (1)

The Close operation (B_CLOSE) 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 (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.

Note: When you close a file inside a transaction, the MicroKernel Engine continues to keep an open handle on the file until the transaction is either aborted or ended so that updates to that file can be handled properly. The position block for the file is no longer available to your application, however.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 succeeds, the position block for the closed file is no longer valid.

If the Close operation fails, the file remains open and the MicroKernel Engine 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 (B_CONTINUOUS) allows you to perform system backups without closing active MicroKernel Engine 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 Engine automatically rolls the delta file changes into the backed up files when those files are taken out of continuous operation mode.

Note: This operation is available only to applications running on a local engine. A client application cannot use this operation for files that are located on a remote machine.

This operation also allows you to safely copy a file while that file is still active. In a client/server set up, the client that begins Continuous Operation on a file must be the client that stops Continuous Operation on that file.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 0 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 path name, excluding only the server name. Separate the names with commas and terminate the list with a binary 0.

The following example is for Windows servers:

```
f:\acct\march.mkd,f:\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 file names 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 Engine does not consider the absence of file names in the data buffer to be an error. If it finds no file names, the MicroKernel Engine takes no action on the Continuous Operation operation.
- The presence of duplicate file names in the data buffer does not affect how the Continuous Operation operation works. The MicroKernel Engine places the specified file in continuous operation mode only once.
- In the same directory, no two files should share the same file name and differ only in their file name extension. For example, a data file named Invoice.btr and another one named Invoice.mkd must not exist in the same directory. This restriction applies because the database engine uses the file name for various areas of functionality while ignoring the file name extension. With continuous operations, the name of the delta file uses the name of the corresponding with ".^^^" for the file name extension. The MicroKernel Engine would attempt to write to the same delta file for both files, possibly causing data corruption or status 85. Also, no files are placed into continuous operations when this condition occurs, even if the files are part of a larger list to be placed into continuous operations.

- 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 the Relational Engine. Files related by referential integrity constraints should be passed in on a single continuous operations call.

  The MicroKernel Engine 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.

The Btrieve API 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 Engine rolls changes from the delta file into the data file, users can continue to update, insert, and read the MicroKernel Engine file just as they normally would. The MicroKernel Engine 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 Engine originally assigned as the owner of the affected files may have been reassigned to another application. Because the MicroKernel Engine 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 Engine is rolling the changes from a delta file into the file, then the MicroKernel Engine rolls all changes into the file when it is first opened after the system is rebooted.

Result

If the Continuous Operation operation succeeds, the MicroKernel Engine returns status code 0, but it returns no values either in the data buffer or in the data buffer length parameter.

If the operation fails, the MicroKernel Engine returns one of the following status codes:

11 The specified file name is invalid.

12 The MicroKernel Engine cannot find the specified file.

41 The MicroKernel Engine 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 Engine 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 file name.

Positioning

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

Create (14)

The Create operation (B_CREATE) generates a new data file from within your application. The Create operation also has subfunctions that allow you to delete or rename a file (see Delete and Rename Subfunctions for the Create Operation).

Note: In the same directory, no two files should share the same file name and differ only in their file name extension. For example, do not name a data file Invoice.btr and another one Invoice.mkd in the same directory. This restriction applies because in some cases the database engine uses only the file name while ignoring the file name extension. In these instances, files that differ only in their file name extension appear identical to the database engine.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	Key Buffer	Key Number
Sent	**X**		**X**	**X**	**X**	**X**
Returned						

Prerequisites

If you are creating an empty file over an existing file, the existing file must be closed before executing the Create operation.

Procedure

1. Set the operation code to 14.
2. Set the file and key specifications and any alternate collating sequences in the data buffer as described in Details. All file and key specification values must be in binary format.
3. Set 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. Set the path name for the file in the key buffer. Terminate the path name with a blank or binary zero. Path names can be up to 255 characters long, including volume name and terminator.

   For more information, see Network Path Formats Supported by Zen Requesters in *Getting Started with Zen*. See also Database URIs in *Zen Programmer's Guide*.
5. Set a value for the key number parameter, using one of the choices in Table .

Details

The data buffer contains specifications for the file and keys to be created. Both Create (14) and Stat (15) use the same data buffer structure, so they are documented together here with slight differences noted. The following tables show the structure of the information for the BTRV and BTRVEX type entry points, both of which are described under Btrieve API Functions.

Note that the order of elements in the specification differs between the two types of entry points. For details, see File Specification Block and Key Specification Block. As shown in the following tables, the data buffer contains a file specification, followed by zero or more key specification blocks, followed by zero or more ACS blocks.

File Specification for BTRV Type Entry Points Used by Create and Stat

File Specification for BTRV Type Entry Points Used by Create and Stat	Data Type[1]	Byte #
Logical fixed record length, combined size of all record fields[3]	Short Int[3]	0–1
Page size. See Page Size.	Short Int[3]	2–3
Number of keys (indexes)	Byte	4
File version.[4] See File Version.	Byte	5
Record count. For Create (14) operations, initialize to 0 to maintain backward compatibility.	Int[3]	6–9
File flags. Set file attributes. See File Flags.	Short Int[3]	10–11
Number of extra pointers. For Create (14), the number of duplicate pointers to reserve for future keys. For Stat (15), the number of remaining pointers. Used with Reserve Duplicate Pointers flag.	Byte	12
Physical page size. Used when compression flag is set. Value is the number of 512-byte blocks.	Byte	13
Preallocated pages. Number of pages to preallocate. Used with page preallocation. For Stat (15), returns number of unused empty pages.	Short Int[3]	14–15

[1]Unless specified otherwise, all data types are unsigned.

[2]For files with variable-length records, the logical record length refers only to the fixed-length portion of the record.

[3]Integers must be stored in little-endian byte order, from low to high.

[4]Returned as 0 from Stat(15) when key number is 0.

File Specification for BTRVEX Type Entry Points Used by Create and Stat

File Specification for BTRVEX Type Entry Points Used by Create and Stat	Data Type[1]	Byte #
Logical fixed record length, combined size of all record fields.	Short Int[3]	0–1
Page size. See Page Size.	Short Int[3]	2–3
File flags. Set file attributes. See File Flags.	Short Int[3]	4–5
Reserved. Initialize to 0 to maintain backward compatibility.	Short Int[3]	6–7
Record count. For Create (14) operations, initialize to 0 to maintain backward compatibility.	Long Long Int[3]	8–15
Number of keys (indexes)	Short Int[3]	16–17
File version.[4] See File Version.	Short Int[3]	18–19
Number of extra pointers. For Create (14), the number of duplicate pointers to reserve for future keys. For Stat (15), the number of remaining pointers. Used with Reserve Duplicate Pointers flag.	Byte	20
Physical page size. Used when compression flag is set. Value is the number of 512-byte blocks.	Byte	21
Preallocated pages. Number of pages to preallocate. Used with page preallocation. For Stat(15), returns number of unused empty pages.	Short Int[3]	22–23
Reserved. Initialize to 0 to maintain backward compatibility.	Long Long Int[3]	24–31

[1]Unless specified otherwise, all data types are unsigned.

[2]For files with variable-length records, the logical record length refers only to the fixed-length portion of the record.

[3]Integers must be stored in little-endian byte order, from low to high.

[4]Returned as 0 from Stat(15) when key number is 0.

File Specification Block

Store the file specification in the first 16 or 32 bytes of the data buffer, numbered beginning with 0. Store information for record length, page size, and number of indexes as integers.

Logical Fixed Record Length

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

Page size is determined by your file format version. See Choosing a Page Size in *Zen Programmer's Guide*. The following table gives page size examples for different file format versions.

Description	File Format Versions	Data Type[1]	Byte #	Example Value[2]
Page Size	6.x through 9.0	Short Int[3]	2–3	512
	6.x through 9.5	Short Int[3]	2–3	1024
	6.x through 9.0	Short Int[3]	2–3	1536
	6.x through 9.5	Short Int[3]	2–3	2048
	6.x through 9.0	Short Int[3]	2–3	3072
	6.x through 9.0	Short Int[3]	2–3	3584
	6.x and later	Short Int[3]	2–3	4096
	9.0 and later	Short Int[3]	2–3	8192
	9.5 and later	Short Int[3]	2–3	16384
	A minimum size of 4096 bytes works best for most files. If you want to fine-tune this, see Creating a File with Page Level Compression in *Zen Programmer's Guide*. When you create a file in 9.5 format or later, if the logical page size specified is not valid for the file format, the MicroKernel rounds the specified value to the next higher valid value if one exists. For all other values and file formats, the operation fails with status code 24. No rounding is done for older file formats.			

[1]Unless specified otherwise, all data types are unsigned.

[2]For simplification, the nonnumeric example values are for C applications.

[3]Integers must be stored in little-endian byte order, from low to high.

Record Count

The number of records in the file. This value is returned by Stat (15). For Create (14), set this field to 0.

Number of Keys

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 Version

The MicroKernel Engine file version to be created. In earlier releases, the MicroKernel Engine used a two-byte integer to receive the number of indexes on a create operation. The high-order byte of this integer was always 0 because the maximum number of indexes is 119. This high-order byte has historically been used in the Stat (15) operation to return the file version, but it can now be used to specify the Create file version without causing errors in previous applications. The supported file versions for Create are 6.0, 7.0, 8.0, 9.0, 9.5, and 13.0, which are represented in the specified byte with hex values 0x50, 0x60, 0x70, 0x80, 0x90, 0x95, and 0xD0, respectively. The following table lists file version flag values for different file format versions.

Description	Data Type[1]	Byte #	Example Value[2]	
File Version	BTRV type: Byte BTRVEX type: Short Int	5 18–19	Version 6.0	0x60
			Version 7.0	0x70
			Version 8.0	0x80
			Version 9.0	0x90
			Version 9.5	0x95
			Version 13	0xD0
			Use database engine default	0x00

[1]Unless specified otherwise, all data types are unsigned.

[2]For simplification, the nonnumeric example values are for C applications.

Number of Extra Pointers

For Create (14), the number of duplicate pointers to reserve for future keys. For Stat (15), the number of remaining pointers. Used with Reserve Duplicate Pointers flag. When this flag is not used, set this field to 0.

Physical Page Size

Used when the Page Compression file flag is set. If the Page Compression flag is not specified, set this field to 0. This field was previously marked as unused.

In data files version 6.x and later, logical pages map to physical pages, stored in a Page Allocation Table (PAT). A physical page is exactly the same size as a logical page. Page compression can be used with file format 9.5 and later. Database pages are compressed at the page level. Each logical page is compressed into one or more physical page units. These individual physical pages are smaller in size than a logical page.

The physical page size field can be used to specify the physical page size to be used for the file. The value specified in this field is multiplied by 512 to determine the actual physical page size used. If 0 is specified, the engine uses a default value of 512 bytes for the physical page size.

The value specified for the physical page size cannot be larger than the value specified for the logical page size. If it is then the engine will round down the value specified for the physical page size so that it is the same as the logical page size. The logical page size needs to be an exact multiple of the physical page size. If it is not then the logical page size is rounded down so that it becomes an exact multiple of the physical page size. If, as a result of these manipulations, the logical and physical values end up to be the same, then page level compression will not turned on for this file. See also Creating a File with Page Level Compression in *Zen Programmer's Guide*

File Flags

The bit settings in the File Flags word specify file attributes. The following table shows the binary, hexadecimal, and decimal representations of 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

Attribute	Constant	Binary	Hex	Decimal
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	4608
Key Number	SPECIFY_KEY_NUMS	0000 0100 0000 0000	400	1024
Use VATs	VATS_SUPPORT	0000 1000 0000 0000	800	2048
Use Page Compression[2]	PAGE_COMPRESSED	0010 0000 0000 0000	2000	8192
Include System Data v2[3]	INCLUDE_SYSTEM_DATA2	0100 0000 0000 0000	4000	16384

[1]If you do not specify whether to include system data in the file, the Btrieve API uses the current System Data setting in the server compatibility properties.

[2]Used only with page-level compression. Used in conjunction with the Physical Page Size key specification. See Creating a File with Page Level Compression in *Zen Programmer's Guide*.

[3]With this flag value, also specify NO_INCLUDE_SYSTEM_DATA for key-only files.

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 Engine 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 Engine ignores the variable length records bit.

The database engine automatically uses data compression on files that use system data and have a record length greater than the maximum length allowed. See Record Length in *Zen Programmer's Guide*.

Set the duplicate pointers bit if you anticipate adding an index sometime *after* creating a file, and if that index has duplicate values and will not be marked as repeating-duplicatable. Setting this bit causes the MicroKernel Engine 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 Engine 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 Engine does not require consecutive key numbers, and files can have gaps between key numbers. When a key is created, by default the MicroKernel Engine 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.

Preallocated Pages

You can specify the number of pages to preallocate. Use this element only if you specified the page preallocation file flag. For more information, see Page Preallocation in *Zen Programmer's Guide*.

Key Specification Block

Place zero or more key specification blocks immediately after the file specification. Allocate a 16- or 24-byte key specification block for each key segment in the file. Store the information for the key position and the key length as integers.

As shown in the following table, the maximum number of key segments allowed depends on the file page size and file format.

Page Size (bytes)	Maximum Key Segments by File Version			
	8.x and earlier	9.0	9.5	13.0
512	8	8	Rounded up[2]	Rounded up[2]
1024	23	23	97	Rounded up[2]
1536	24	24	Rounded up[2]	Rounded up[2]
2048	54	54	97	Rounded up[2]
2560	54	54	Rounded up[2]	Rounded up[2]
3072	54	54	Rounded up[2]	Rounded up[2]
3584	54	54	Rounded up[2]	Rounded up[2]
4096	119	119	204[3]	183[3]
8192	n/a[1]	119	420[3]	378[3]
16384	n/a[1]	n/a[1]	420[3]	378[3]

[1]"n/a" means "not applicable"

[2]"Rounded up" means that the page size is rounded up to the next size supported by the file version. For example, 512 is rounded up to 1024, 2560 is rounded up to 4096, and so forth.

[3]While a 9.5 format or later file can have more than 119 segments, the number of indexes is limited to 119.

See also the status codes 26: The number of keys specified is invalid and 29: The key length is invalid, both in *Status Codes and Messages*.

The following table specifies the data buffer structure of key segment for a Create (14) or Stat (15) operation. Each key specification block is 16 bytes for BTRV type entry points or 24 bytes

for BTRVEX types. Where two data types are given, but first is used with BTRV and the second with BTRVEX. The offsets repeat for each key block.

Description	Data Type[1]	Offset for BTRV	Offset for BTRVEX
Key Position. Position of the first byte of the key within the record. The first byte in the record is 1.	Short Int[2]	0–1	0–1
Key Length. Length of the key in bytes.	Short Int[2]	2–3	2–3
Key Flags. Key attributes.	Short Int[2]	4–5	4–5
Reserved. Not used for Create (14). Initialize to 0 to maintain backward compatibility.	Short Int[2]	—	6–7
Unique keys. Not used for Create (14). Initialize to 0 to maintain backward compatibility.	Int or Long Long Int[2]	6–9	8–15
Extended Data Type. Used if the key flags specify use of extended data types.	Byte or Short Int[2]	10	16–17
Null Value (legacy nulls only). Used if the key flags specify Null Key (All Segments) or Null Key (Any Segment). An exclusion value for the key. See Null Value for more conceptual information on legacy nulls and true nulls.	Byte	11	18
Reserved. Not used for Create (14). Initialize to 0 to maintain backward compatibility.	Short Int[2] or Byte	12–13	19
Manually Assigned Key Number. Key number used if the file attributes specify one.	Byte or Short Int[2]	14	20–21
ACS Number. ACS number. Used if the key flags specify Use Default ACS, Use Numbered ACS in File, or Use Named ACS.	Byte	15	22
Reserved. Not used for Create (14). Initialize to 0 to maintain backward compatibility.	Byte	—	23

[1]Unless specified otherwise, all data types are unsigned.

[2]Integers must be stored in little-endian byte order, from low to high.

Key Position

The key position is the byte offset at which the key or key segment begins. Positions are relative to 1. A key at the beginning of the record starts at position 1. There is no position 0.

Key Length

The length of the key or key segment. Maximum key length, including all segments, is 255 bytes.

Key Flags

The bit settings in the key flag word specify key attributes. The following table shows the binary, hexadecimal, and decimal representations of key flag values.

Attribute	Constant	Binary	Hex	Decimal
Duplicates allowed (linked duplicates is default, or combine with REPEAT_DUPS_KEY for repeating duplicates)	DUP	0000 0000 0000 0001	0x1	1
Modifiable Key	MOD	0000 0000 0000 0010	0x2	2
Use Old Style BINARY Data Type	BIN	0000 0000 0000 0100	0x4	4
Use Old Style STRING Data Type (bits 2 and 8 must be 0)		0000 0000 0000 0000	0x0	0
Null Key (All Segments)	NUL	0000 0000 0000 1000	0x8	8
Segmented Key	SEG	0000 0000 0001 0000	0x10	16
Use Default ACS	ALT	0000 0000 0010 0000	0x20	32
Use Numbered ACS in File	NUMBERED_ACS	0000 0100 0010 0000	0x420	1056
Use Named ACS	NAMED_ACS	0000 1100 0010 0000	0xC20	3104
Descending Sort Order	DESC_KEY	0000 0000 0100 0000	0x40	64
Repeating Duplicates, used with DUP	REPEAT_DUPS_KEY	0000 0000 1000 0000	0x80	128
Use Extended Data Type	EXTTYPE_KEY	0000 0001 0000 0000	0x100	256
Null Key (Any Segment)	MANUAL_KEY	0000 0010 0000 0000	0x200	512
Case Insensitive Key	NOCASE_KEY	0000 0100 0000 0000	0x400	1024

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, sum their values. For example, if the key is an extended type, part of a segmented key, and collated in descending order, then initialize the key flag 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, Repeating Duplicates, Modifiable Key, and Null Key values. If you specify the legacy Null Key attribute, either All Segments or Any Segment, then you can assign different null values for individual segments.
- All segments of the same key must use the same alternate collating sequence (ACS).
- Individual segments of the same key can have different Descending Sort Order and Extended Data Type values.

The ACS applies only to keys of type STRING, LSTRING, ZSTRING, WSTRING, and WZSTRING. 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, segments that designate an ACS sort by that ACS, while those with no ACS sort according to their respective types.

Extended Data Type

You specify the Extended Data Type byte of the key specification block as a binary value. The following table shows the codes for the extended data types.

Type	Code
STRING	0
INTEGER	1
FLOAT	2
DATE	3
TIME	4
DECIMAL	5
MONEY	6
LOGICAL	7
NUMERIC	8

Type	Code
BFLOAT	9
LSTRING	10
ZSTRING	11
UNSIGNED BINARY	14
AUTOINCREMENT	15
NUMERICSTS	17
NUMERICSA	18
CURRENCY	19
TIMESTAMP	20

Type	Code
CLOB	21
WSTRING	25
WZSTRING	26
GUID	27
AUTOTIMESTAMP	32
TIMESTAMP2	34
NULL INDICATOR SEGMENT	255

Extended data type codes 12, 13, 16, and 22–24 are reserved for future use. Note that the CLOB type is included for Get Extended operations but cannot be used to create an index.

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 API, while allowing newer applications to use extended data types exclusively.

Regarding the data type you assign to a key, the MicroKernel Engine does not ensure that the records entered follow 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 API application would succeed, but an ODBC application that tries to access the same data using the ODBC TIMESTAMP format might fail, since 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, see Btrieve Key Data Types in *SQL Engine Reference*.

Null Value

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 Engine to recognize as the null value for each key segment. This is in reference to the legacy null and does not reflect true nulls. For a discussion of null support, see Null Value in *Zen Programmer's Guide*.

Manually Assigned Key Number

The MicroKernel Engine 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 the number of each key as a binary value in 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 page size of the file).

The ability to manually assign key numbers complements to the ability to delete a key and not have the MicroKernel Engine 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 Engine 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

For keys that use a specific ACS, 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

In the data buffer for a Create operation, collating sequences appear one after another immediately following the last key specification block. The 265 bytes used to specify an ACS, ISR, or ICU collation are described in the following tables.

User-Defined ACS Files

To create an ACS that sorts string values differently from the ASCII standard, your application must place 265 bytes directly into the following data buffer.

Offset	Length (Bytes)	Description
0	1	Signature byte. Specify 0xAC.
1	8	A unique 8-byte name that identifies the ACS to the MicroKernel Engine.
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 ACS files, see Alternate Collating Sequences in *Zen Programmer's Guide*.

International Sort Rules (ISRs)

To specify an ISR table name, your application must place 265 bytes directly into the following data buffer.

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

Unicode Collations

To specify a Unicode collation according to the International Components for Unicode (ICU) standard, your application must place 265 bytes directly into the following data buffer.

Offset	Length (Bytes)	Description
0	1	Signature byte. Specify 0xAE.
1	16	Name of the supported ICU collation, either u54-msft_enus_0 or root. You must fill with spaces to 16 bytes.
17	248	Filler.

Data Buffer Length

The data buffer must be long enough to include the file specifications, the key specifications, and any ACS files that have been defined. Do *not* specify the file record length in this parameter.

For example, to create a file using the BTRV entry point 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	+	Key Spec	+	Key2 Spec	+	ACS	
16	+	16	+	16+	+	265	= 313

Key Number

The key number for the Create operation determines whether the MicroKernel Engine warns you when a file of the same name already exists, and also whether the MicroKernel Engine should use a local or remote engine when creating the file.

Use the following table to choose the value for the key number parameter.

CREATE Operation	No preference	Force local engine to create file	Force remote engine to create file
Normal create (overwrite if file already exists)	0	6	99
Return Status 59 if file already exists	–1	7	100

Delete and Rename Subfunctions for the Create Operation

The Create operation has two additional subfunctions that you can use to delete or rename files.

Before Pervasive.SQL v8.5, it was possible to manipulate MicroKernel Engine files through the operating system because the engine depended on the rights and privileges given to the Zen user by the operating system.

After the introduction of Zen database security in v8.5, these operating system access rights might be removed when a database is secured against unauthorized access. The options for programmatically deleting or moving a file may change because operating system rights are not always available.

The rename and delete subfunctions are implemented as Create operations with alternate key numbers. You do not need to provide a file specification as you do when creating a new data file. The following table shows how to set up the Create operation to use the rename or delete subfunctions.

Function	Key Number to Use	Description	Place in Data Buffer	Place in Key Buffer
Rename File	–127	Rename an existing file in the data buffer to the name in the key buffer	Existing File Name	New File Name
Delete File	–128	Delete a file	N/A	Existing File Name

These subfunctions have been modified to work with the security model in that they will accept a URI in place of a file name in the key buffer and data buffer, if needed, to indicate a MicroKernel Engine file to delete or rename. This allows you to provide security information with the operation. For details about URI connection strings, see Database URIs in *Zen Programmer's Guide*.

The security information is processed like a normal Create or Open operation. The user must be authenticated and have DB_RIGHT_CREATE, DB_RIGHT_ALTER and DB_RIGHT_OPEN privileges for the existing files and for the directory where the new file will be located if applicable.

For the following Create subfunctions, an X indicates valid URI parameters in key buffers.

Function	URI parameter file=	URI parameter dbfile=	URI parameter table=
Rename	**X**	**X**	
Delete	**X**	**X**	

For the following Create subfunctions, an X indicates valid URI parameters in data buffers.

Function	URI parameter file=	URI parameter dbfile=	URI parameter table=
Rename	**X**	**X**	
Delete	Not applicable	Not applicable	Not applicable

Notes on Rename and Delete Subfunctions

- The RenameFile and DeleteFile subfunctions cannot be used on files that are bound to specific databases because they do not affect the contents of the miscellaneous page.
- If a file contains an owner name, the owner name check is not performed by the new subfunctions. The owner name is still needed to view the contents of the files.

Result

If the Create operation succeeds, the MicroKernel Engine 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 fails, the MicroKernel Engine 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.
41	The MicroKernel does not allow the attempted operation. (File format is lower than version 13.00.)
48	The alternate collating sequence definition is invalid.
49	The extended data type is invalid.
59	The specified file already exists.

104 The MicroKernel Engine does not recognize the locale.

105 The file cannot be created with Variable-tail Allocation Tables (VATs).

134 The MicroKernel Engine 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 (B_BUILD_INDEX) adds a key to an existing file.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 page size of the file. The following table lists these values.

Page Size (bytes)	Maximum Key Segments by File Version			
	8.x and earlier	9.0	9.5	13.0
512	8	8	Rounded up[2]	Rounded up[2]
1024	23	23	97	Rounded up[2]
1536	24	24	Rounded up[2]	Rounded up[2]
2048	54	54	97	Rounded up[2]
2560	54	54	Rounded up[2]	Rounded up[2]
3072	54	54	Rounded up[2]	Rounded up[2]
3584	54	54	Rounded up[2]	Rounded up[2]
4096	119	119	204[3]	183[3]
8192	n/a[1]	119	420[3]	378[3]
16384	n/a[1]	n/a[1]	420[3]	378[3]

Page Size (bytes)	Maximum Key Segments by File Version			
	8.x and earlier	9.0	9.5	13.0

[1]"n/a" means "not applicable"

[2]"Rounded up" means that the page size is rounded up to the next size supported by the file version. For example, 512 is rounded up to 1024, 2560 is rounded up to 4096, and so forth.

[3]While a 9.5 format or later file can have more than 119 segments, the number of indexes is limited to 119.

See also the status codes 26: The number of keys specified is invalid and 29: The key length is invalid, both in *Status Codes and Messages*.

- 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 key specification block in the data buffer. Use the same structure documented under Create (14). 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 the size of one key specification block and initialized to zeros.
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. 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. 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:

ACS Type	Length (Bytes)	Description
User-defined ACS	1	Signature 0xAC
"	8	ACS table name
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 or 24) * (# 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 or 24) * (# 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 a system key (also called system data or system data v2), specify 0xFD (that is, key number 125 + 128) or 0XFC (key number 124 + 128). Note that for BTRVEX this bias results in a small positive value and should not be sign-extended.

Note: Key numbers must be unique to the file. They must also be valid. The value of each key number must be less than the maximum number of key segments allowed for the specified page size.

Details

The MicroKernel Engine 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 Engine 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 Engine 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 Engine first checks for an existing ACS (using the name you specified) before adding it to the file. If the MicroKernel Engine finds an

existing ACS with the name you specified, the MicroKernel Engine 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 Engine 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 Engine client and one of the clients starts a Create Index process, remote clients can perform Get and Step operations on the open file while the MicroKernel Engine client 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 Engine 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, then the MicroKernel Engine 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 Engine returns status code 84 if a client tries to create an autoincrement key while another client has locked a record. Similarly, the MicroKernel Engine 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 Engine 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 succeeds, 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 fails, the MicroKernel Engine 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 Engine can return the following status codes:

22	The data buffer length is too short.
27	The key position is invalid.
41	The MicroKernel Engine 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 Engine does not recognize the locale.
134	The MicroKernel Engine cannot read the International Sorting Rule.
135	The specified International Sort Rule table is corrupt or otherwise invalid.
136	The MicroKernel Engine 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 other keys in the file. However, the MicroKernel Engine 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 (32) and reissue Create Index.

Positioning

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

Delete (4)

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

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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.

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 may not be a valid operation if performed immediately after an extended Get or extended Step operation because the current record is unpredictable.

After you perform a Delete operation, a later Get Next or Get Previous operation must use the same key number and key buffer as the last operation that established logical position. If you use another value, the MicroKernel Engine returns status code 7.

The MicroKernel Engine does not allow Delete operations after a Get Key (+50). Before the MicroKernel Engine 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 Engine 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 Engine cannot perform its passive concurrency conflict checking without the compare. When the Delete fails, the MicroKernel Engine returns status code 8.

Result

If the Delete operation succeeds, the MicroKernel Engine 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 fails, the MicroKernel Engine returns one of the following status codes:

8	The current positioning is invalid.
80	The MicroKernel Engine encountered a record-level conflict.
84	The record or page is locked.
85	The file is locked.

Delete operations never reduce file size. Empty Space created by deletion of records is reused when records are added in the future. Disk space can be recovered only by recreating the file and inserting all of the records into it. The Rebuild and Defragmenter utilities can help accomplish this recovery.

Positioning

The Delete operation destroys the complete physical location information and the logical current record position but does not change the physical and logical positions of either the next record or the previous record.

Drop Index (32)

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

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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. To drop the second system key for system data v2, specify 124.

Details

If you drop a system key, you can rebuild it using Create Index (31).

When you delete a key, the MicroKernel Engine automatically renumbers all higher-numbered keys, unless you specify otherwise. The MicroKernel Engine 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 Engine renumbers the keys as 1 and 6.

If you do not want the MicroKernel Engine to automatically renumber keys, add a bias of 128 to the value you supply for the key number parameter. This bias allows you to leave gaps in the key numbering, and 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 Create Index (31), which allows you to specify a key number.

However, if you delete a key without renumbering higher-numbered keys and then a user clones the affected file without assigning specific key numbers, the cloned file has different key numbers from the original.

Note: Users can clone files using the Btrieve Maintenance tool or butil, its command line version. Cloning creates a new, empty file with the same statistics as an existing file.

Result

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

If the Drop Index operation fails, the MicroKernel Engine returns one of the following status codes:

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

If processing is interrupted while the MicroKernel Engine is dropping an index, you can access the data in the file by the file's other keys. The MicroKernel Engine 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 (B_END_TRAN) completes a transaction and makes the appropriate changes to the data files. It also unlocks all files and records locked by the transaction.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	Key Buffer	Key Number
Sent	**X**					**X**
Returned						

Prerequisites

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

Procedure

Set the operation code to 20. While the MicroKernel Engine 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 succeeds, 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 fails, the MicroKernel Engine returns the following status code:

38 The MicroKernel Engine 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 (B_GET_PERCENT) is one of two Btrieve API operations that window-oriented applications can use to implement scroll bars. The other is Get By Percentage (44). Find Percentage returns the approximate position of a record either relative to a key path or as the physical location of the record within the file. The position is expressed as a percentage value. See Result for a definition of the range of percentage values.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 the physical location of a record 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 the physical location of a record in the file, you must provide that physical location in the data buffer parameter. You can retrieve this location with Get Position (22). Be consistent in the use of BTRV or BTRVEX type entry points.

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 physical location of a record in the file, store the record's 4- or 8-byte physical address in the data buffer. If you are seeking the percentage relative to the record's key path and wish to specify a granularity for the search, set your data buffer parameter as specified in Granularity. 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 or 8 bytes. This 4-byte minimum is a requirement of the MicroKernel Engine internal implementation. If specifying a granularity for the search, set the data buffer length to a minimum of 12 or 16 bytes.
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:
    - If you are seeking the percentage by a key path, set the key number parameter to the actual key number.
    - If you are seeking the percentage by the physical record location, set the key number parameter to –1.

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 Engine 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.

Granularity

The granularity setting is optional and allows you to choose the factor by which the percentage is measured. In releases before Zen 9, this value was always 10000.

If you want to specify the granularity, follow these steps:

To specify a granularity in the Find Percentage operation

1. Place the signature ExPc in the 4 bytes of the data buffer (0x45, 0x78, 0x50, 0x63) after the record address area.

2. Place the desired granularity in the 4 bytes after the signature as a LoHi Intel integer. The granularity you choose can be any number from 1 to 0xFFFFFFFF.
3. Ensure that your data buffer length is at least 12 or 16 bytes, depending on the entry point used.

The following table summarizes positions and layouts for these steps.

	BTRV, BTRVID, BTRCALL, BTRCALLID	BTRVEX, BTRVEXID
Record address	4 bytes, offset 0	8 bytes, offset 0
Signature granularity	4 bytes, offset 4	4 bytes, offset 8
General granularity	4 bytes, offset 8	4 bytes, offset 12
Total size	4 or 12 bytes	8 or 16 bytes

For example, if you want to get the 100th record from a file that contains 365 records, you can use Find Percentage (45) with 100 as the percentage and 365 as the granularity.

Result

If the Find Percentage operation succeeds, the MicroKernel Engine returns the relative 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 10000 (100.00 percent). Note this is not the physical or logical position.

The percentage value is returned as a 4-byte integer in low-byte, high-byte order. For example, when using default granularity:

Returned Value Hex	Returned Value Dec	Percentage in Key Path or File
88h 13h	5000	50%

The MicroKernel Engine also returns a data buffer length of at least 4 if the operation succeeds.

If the Find Percentage operation fails, the MicroKernel Engine 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 Engine does not allow the attempted operation.
43	The specified record address is invalid.
82	The MicroKernel Engine lost positioning.

Positioning

The Find Percentage operation does not change any currency information.

Get By Percentage (44)

The Get By Percentage operation (B_SEEK_PERCENT) is one of two Btrieve API operations that can be used by window-oriented applications for implementing scroll bars. The other is the Find Percentage (45). Get By Percentage retrieves a record by the relative position of the record 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 actual physical location of the record in the file.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations Guide*.
2. Pass the position block for the file.

3. Store the percentage value as a 4-byte integer in the data buffer. See Details 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 Engine internal implementation requires that you set the data buffer length value to a minimum of 4 bytes). If specifying a granularity for the search, set the data buffer length to a minimum of 12 bytes.

5. Set the key number parameter.

    - 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. To use the second system key for system data v2, specify 124.
    - If you are seeking the record by the physical record position in the file, set the key number parameter to –1.

Details

If you are not specifying a granularity (see Granularity), the range of acceptable percentage values for the first two bytes of the data buffer parameter is from 0 (indicating the beginning of the key path or file) through 10000 (the end of the key path or file). The value corresponds to a range of 0% to 100.00%, assuming two implied decimal places. If you want to find the record approximately 33.33% through the file, pass the value 3333 in the data buffer. 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 5000 (0x1388). After byte-swapping 0x1388, store 0x88 and 0x13 in the first two bytes of the data buffer parameter.

If you wish to specify a granularity for the search, set your data buffer parameter as specified in Granularity.

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 Engine 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.

Granularity

The granularity setting is optional and allows you to choose the factor by which the percentage is measured. In releases before Zen 9, this value was always 10000.

If you want to specify the granularity, follow these steps:

To specify a granularity in the Get By Percentage operation

1. Place the signature ExPc in the second 4 bytes of the data buffer (0x45, 0x78, 0x50, 0x63) after the percentage.
2. Place the desired granularity in the 4 bytes after the signature as a LoHi Intel integer. The granularity you choose can be any number from 1 to 0xFFFFFFFF.
3. Ensure that your data buffer length is at least 12 bytes.

For example if you want to get the 100th record from a file that contains 365 records, you can use Get By Percentage (44) with 100 as the percentage and 365 as the granularity.

Result

If the Get By Percentage operation succeeds, the MicroKernel Engine 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 Engine 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 Engine 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 Engine does not return any information in the key buffer parameter.

Note: When Get By Percentage is seeking a record relative to a key path, and the key contains duplicate values, the MicroKernel Engine always returns the first record containing the duplicated value. This implementation detail can affect the accuracy of the operation.

If the Get By Percentage operation fails, the MicroKernel Engine 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 Engine does not allow the attempted operation.
43	The specified record address is invalid.
82	The MicroKernel Engine 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 physical location of the record in the file, the Get By Percentage operation establishes the new physical currency based on the retrieved record.

If the Get By Percentage operation fails, the MicroKernel Engine changes no currency.

Get Direct/Chunk (23)

The Get Direct/Chunk operation (B_GET_DIRECT) can retrieve one or more portions, called *chunks*, of a record. This operation is especially useful on files containing records longer than the maximum data buffer size. 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

	Op Code	Pos Block	Data Buf	Data Buf Len	Key Buffer	Key Number
Sent	**X**	**X**	**X**	**X**		**X**
Returned		**X**	**X**	**X**		

Prerequisites

- The file must be open.
- You must provide the physical location of the record. You can retrieve this location with Get Position (22). Be consistent in the use of BTRV or BTRVEX type of entry points.
- 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.

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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations 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 or the number of bytes you requested for the MicroKernel Engine 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.

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 (areas containing [*]): 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

Data Buffer for Random Chunk Operations

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

Element	Length (Bytes)	Description
Record Address	4 or 8[1]	The physical location of the record. You can retrieve this location with Get Position (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.
Chunk Definition (Repeat for each chunk)	12 (for 32-bit applications) 16 (for 64-bit applications)	Each Chunk Definition is a 4-byte Chunk Offset, followed by a 4-byte Chunk Length, followed by a 4-byte User Data for 32-bit applications or an 8-byte User Data for 64-bit applications, 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 65535[2]. • User Data – (Used only for indirect descriptors.) For 32-bit applications, a 32-bit pointer to the actual chunk data. For 64-bit applications, a 64-bit pointer to the actual chunk data. The MicroKernel Engine ignores this element for direct chunk descriptor subfunctions.

[1]Size depends on whether you use BTRV or BTRVEX type entry points.

[2]For BTRVEX, chunk size is limited to 65535, but multiple chunks can be returned in a large data buffer.

The following table shows a sample data buffer for a 32-bit application for fetching direct random chunks using a BTRV entry point.

Element	Sample Value	Length (Bytes)
Record Address	0x00000628	4
Subfunction	0x80000000	4
NumChunks	3	4
Chunk 0		
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 (areas containing [*]): 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.

Data Buffer for Rectangle Chunks

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.

Element	Length (Bytes)	Description
Record Address	4 or 8[1]	The 4-byte physical location of the record. You can retrieve this location with Get Position (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 65535[2].
Distance Between Rows	4	Number of bytes between the beginning of each chunk.

Element	Length (Bytes)	Description
User Data	4 (for 32-bit applications) 8 (for 64-bit applications)	(Used only with indirect descriptors.) For 32-bit applications, a 32-bit pointer to the location into which the MicroKernel Engine stores bytes after retrieving them from each row. For 64-bit applications, a 64-bit pointer to the location into which the MicroKernel Engine stores bytes after retrieving them from each row. The MicroKernel Engine 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 application memory, at the address specified by User Data. The MicroKernel Engine ignores this element for direct rectangle descriptors. However, you must still allocate the element and initialize it to 0.

[1]Size depends on whether you use BTRV or BTRVEX type entry points.

[2]For BTRVEX, chunk size is limited to 65535, but multiple chunks can be returned in a large data buffer.

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 Engine uses the descriptor when copying the returned chunks to the locations specified by User Data elements. If you overwrite your chunk descriptor, the MicroKernel Engine 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 application memory with either more or fewer bytes between rows, Application Distance Between Rows allows you to pass that information to the MicroKernel Engine.

When you use an indirect rectangle descriptor, the MicroKernel Engine 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 Engine stores data from the first row in offset 0 of User Data. The MicroKernel Engine stores the data from the second to an address specified by User Data + Application Distance Between Rows. The MicroKernel Engine stores data from the third row in the address specified by User Data + (Application Distance Between Rows * 2), and so on.

The following table shows a sample data buffer for a 32-bit application for fetching a direct rectangle chunk using a BTRV entry point.

Element Name	Sample Value	Length (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 Engine calculates the subfunction 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 Engine ignores the Offset element in the chunk descriptor.

Result

If the Get Direct/Chunk operation succeeds and a direct chunk descriptor is used, the MicroKernel Engine returns the chunks one after another in the data buffer. If you used an indirect random chunk descriptor, the MicroKernel Engine returns the data to the locations specified by the User Data element of each chunk. If you used an indirect rectangle descriptor, the MicroKernel Engine returns the data to locations it derives from the User Data and Application Distance Between Rows elements.

The MicroKernel Engine 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 Engine returning status code 103), or if the offset and length of any chunk combine to exceed the length of the record. In the latter case, the MicroKernel Engine 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 Engine returns one of these status codes, your application should check the data buffer length parameter return value to see how much data the MicroKernel Engine actually returned.

22 The data buffer parameter is too short.

54 The variable-length portion of the record is corrupt.

103 The chunk offset is too big.

If the MicroKernel Engine 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 Engine 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 (B_GET_DIRECT) 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 Get Position (22) to retrieve the 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 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

	Op Code	Pos Block	Data Buf	Data Buf Len	Key Buffer	Key Number
Sent	**X**	**X**	**X**	**X**		**X**
Returned		**X**	**X**	**X**	**X**	

Note: The key number parameter is not needed when you are performing a Get Direct/Record operation on a data-only file.

Prerequisites

- The file must be open.
- You must provide the 4- or 8-byte physical location of the record. You can retrieve this location with Get Position (22), which returns the physical address of the current record. Be consistent in the use of BTRV or BTRVEX type of entry points.

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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations Guide*.

2. Pass the position block for the file.
3. Store the 4- or 8-byte position of the requested record in beginning of the data buffer. Size depends on whether you are using BTRV or BTRVEX type entry points.
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 Engine to establish logical currency. Specify –1 if you do not want the MicroKernel Engine to establish logical currency. To use the system-defined log key (also called system data), specify 125. To use the second system key for system data v2, specify 124.

Result

If the Get Direct/Record operation succeeds, the MicroKernel Engine 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 fails and the MicroKernel Engine cannot return the requested record, the MicroKernel Engine 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 Engine 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 (B_GET_DIR) returns the current directory for a specified logical disk drive.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 Engine 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 (B_GET_EQUAL) 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

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations 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. If the key consists of multiple segments, make sure you define the key buffer to represent all segments and fill in values for all segments. If you don't have search criteria for all segments, use the GetGreaterOrEqual operation instead.

5. Set the key number to the correct key path. To use the system-defined log key (also called system data), specify 125. To use the second system key for system data v2, specify 124.

Result

If the Get Equal operation succeeds, the MicroKernel Engine 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 fails, the MicroKernel Engine 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.

This operation returns status code 4 if the key contains a nonzero value in a null indicator segment. You cannot use GetEqual to find records that are NULL, because by definition NULL is indeterminate, or not equal to anything. If you need to find NULL values, use GetFirst followed by GetNext.

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 (B_GET_FIRST) 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

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations 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. To use the second system key for system data v2, specify 124.

Result

If the Get First operation succeeds, the MicroKernel Engine 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 fails, the MicroKernel Engine 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 Than (8)

The Get Greater Than operation (B_GET_GT) 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 Than operation on descending keys, the next greater value refers to a value lower than the one specified in the key buffer.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations 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. To use the second system key for system data v2, specify 124.

Result

If the Get Greater Than operation succeeds, the MicroKernel Engine 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 operation fails, the MicroKernel Engine 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 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 (B_GET_GE) 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 Engine 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

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations 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. To use the second system key for system data v2, specify 124.

Result

If the Get Greater Than or Equal operation succeeds, the MicroKernel Engine 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 fails, the MicroKernel Engine 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 Than (8)
- Get Greater Than or Equal (9)
- Get Less Than (10)
- Get Less Than or Equal (11)
- Get First (12)
- Get Last (13)
- Get By Percentage (44)

Parameters

The parameters are the same as those for the corresponding Get operation, except that the MicroKernel Engine 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 Get Equal (5), set the operation code to 55.

The MicroKernel Engine does not allow Delete or Update operations after a Get Key (+50). Before the MicroKernel Engine 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 Engine 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 Engine cannot perform its passive concurrency conflict checking without the compare. When the Update or Delete fails, the MicroKernel Engine returns status code 8.

Result

If the MicroKernel Engine finds the requested key, it returns the key value in the key buffer and status code 0. Otherwise, the MicroKernel Engine 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 Engine 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 Engine does not allow an Update or Delete operation to follow a Get Key operation.

Get Last (13)

The Get Last operation (B_GET_LAST) 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

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations 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. To use the second system key for system data v2, specify 124.

Result

If the Get Last operation succeeds, the MicroKernel Engine 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 fails, the MicroKernel Engine 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 (B_GET_LT) 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

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations 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. To use the second system key for system data v2, specify 124.

Result

If the Get Less Than operation succeeds, the MicroKernel Engine 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 fails, the MicroKernel Engine 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 (B_GET_LE) 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 Engine 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

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations 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. To use the second system key for system data v2, specify 124.

Result

If the Get Less Than or Equal operation succeeds, the MicroKernel Engine 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 fails, the MicroKernel Engine 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 (B_GET_NEXT) 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

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations 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 that established logical position.

Pass the key buffer exactly as the MicroKernel Engine returned it on the previous call, since the information stored there may be needed to determine the current position in the file.

5. Set the key number parameter to the key path used on the previous call that established logical position. You cannot change key paths using a Get Next operation.

Result

If the Get Next operation succeeds, the MicroKernel Engine 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 fails, the MicroKernel Engine 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 Engine 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 Delete Extended (85)

The Get Next Delete Extended operation (B_GET_NEXT_EXT_DELETE) 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 compares the examined record or records to a filter condition and deletes those that match. The filter condition is a logic expression and is not limited to key fields.

As noted under this topic, this operation uses the same input and output buffer structures and returns the result described under Get Next Extended (36). See that operation for more information.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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. You can establish logical positioning by issuing any unextended Get operation, such as a Get Equal.

Procedure

1. Set the operation code to 85.

   By default, the lock bias is no-wait, and any lock bias setting is ignored. Behavior is identical to +500: If the engine cannot delete a locked record, it returns immediately without retrying the operation.

2. Pass the position block for the file.

3. Specify a data buffer to accommodate either the input structure or the returned output, whichever is larger. Initialize the data buffer according to the instructions in Table .

4. Specify the buffer size as either the length of the input structure in Table or the length of the returned output in Table , whichever is larger.

5. Specify the key value from the previous operation in the key buffer that established logical position.

   Pass the key buffer exactly as the MicroKernel Engine returned it on the previous call, since the information stored there may be needed to determine the current position in the file.

6. Set the key number parameter to the key path used on the previous call that established logical position. You cannot change key paths using a Get Next Delete Extended operation.

Details

The following topics under Get Next Extended (36) cover the structure of the extended operation input buffer and use of its filter segment, as well as the structure of the output buffer that returns the result:

- Input Buffer for Extended Operations
- Output Buffer for Extended Operations

Result

If the Get Next Delete Extended operation succeeds, the MicroKernel Engine returns the following:

- In the output buffer, one or more record addresses from one or more records. For details, see Output Buffer for Extended Operations.
- In the output buffer length, the total number of bytes received.
- In the key buffer, the key value for the last data record examined.

A Get Next Delete Extended operation may fail for the same reasons as other Step and Get Extended operations, returning 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 Engine lost positioning.
134	The MicroKernel Engine cannot read the International Sorting Rule.
135	The specified International Sort Rule table is corrupt or otherwise invalid.
136	The MicroKernel Engine cannot find the specified alternate collating sequence in the file.

If the output buffer length is zero, then no records were deleted. However, the operation may have succeeded in deleting some records before failing. The following list gives some examples of these partial successes:

- The output buffer no longer has room to write out the record address for the current record that matches the filter conditions. That record is not deleted, and the operation fails with status code 22.
- Another client has locked the current record, and the operation fails with status code 84.

In these cases, the output buffer length is greater than zero, and the first two bytes of the buffer give a count of the number of deleted records.

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

Positioning

The Get Next Delete Extended operation does not establish currency. However, you can do a Get Next or Get Previous operation, and the next or previous logical position is valid. A valid current position also becomes available by using Get Position (22) and Get Direct/Record (23).

The following list shows the relationships of selected status codes to filter conditions:

- Status 60 (reject count reached): The current position is a record that does not match the filter condition.
- Status 64 (filter limit reached): The current position is a record that may not match the filter condition. Any attempt to step to the next record will not match the filter condition.
- Status 84 (record or page locked): The current position is a record that may not match the filter condition. It is also possible that the next record matched the filter condition but could not be deleted because of locking.

- Status 22 (data buffer full): The current position is a record that matches the filter condition, but the data buffer does not have space to write the record address, so the MicroKernel Engine did not delete the record.
- Status 9 (end of file): The current position is both logically and physically invalid.

Get Next Extended (36)

The Get Next Extended operation (B_GET_NEXT_EXTENDED) 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 compares the examined record or records to a filter condition and retrieves those that match. The filter condition is a logic expression and is not limited to key fields.

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

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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. You can establish logical positioning by issuing any unextended Get operation, such as a Get Equal.

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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations Guide*.
2. Pass the position block for the file.

3. Specify a data buffer to accommodate either the input structure or the returned output, whichever is larger. Initialize the data buffer according to the instructions in Table .
4. Set the buffer size as either the length of the input structure in Table or the length of the returned output in Table , whichever is larger.
5. Set the key value from the previous operation in the key buffer that established logical position. Pass the key buffer exactly as the MicroKernel Engine returned it on the previous call, since the information stored there may be needed to determine the current position in the file.
6. Set the key number parameter to the key path used on the previous call that established logical position. You cannot change key paths using a Get Next Extended operation.

Details

The following topics cover the structure of the extended operation input buffer and use of its filter segment, as well as the structure of the output buffer that returns the result of the operation.

- Input Buffer for Extended Operations
- Collation of LIKE Results
- Using the JSON QUERY Operator
- Processing of Logical AND and OR in a Filter
- Examples of Filtering Records
- Output Buffer for Extended Operations

Input Buffer for Extended Operations

The following table provides the structure of the input buffer for extended Get and Step operations. This buffer applies to all extended operations, with differences in usage as noted.

Element	Length (Bytes)	Description
Header	2	Exact length of the input data buffer.
	2	One of two string constant values (fixed length, do not null-terminate): "EG" – Begin with the record after the one at which you are positioned. "UC" – Begin with the record at which you are positioned.
Filter (Fixed portion)	2	Maximum reject count, the number of records that do not match the filter that the database engine can skip. You can set a value from 0 to 65535. Zero means the engine uses 4095 default.
	2	Number of terms in the logic expression of the filter. Zero means MicroKernel Engine performs no filtering. The only limit to the number of terms is the size of the data buffer. In Pervasive.SQL 2000i SP3 only, the limit for the number of terms is 119.

Element	Length (Bytes)	Description
Filter (Repeat this segment once for each term in the logic expression)	1	Data type of the field. Use one of the codes shown here.
	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 7 – Extended operation code • Add a +8 bias to compare strings using a collation sequence in the file. • Add a +32 bias to compare strings using the first ACS in the file. The +32 bias is ignored if a +8 bias is used. • Add a +64 bias if the second operand is another field of the record. • Add a +128 bias to compare strings without case sensitivity.
	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
	1	This field is present only when the comparison code is 7: 1 – LIKE operator 2 – NOT LIKE operator 3 – JSON QUERY operator
	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. • When comparing a field to a constant: the value of the constant. The length of the constant (*n*) must equal the length of the field. • When the comparison code is 7, this element has this structure: – 2-byte length specifying the size of the LIKE clause, including a null terminator – Null-terminated string containing the LIKE or NOT LIKE clause or JSON query string – The LIKE or NOT LIKE clause uses SQL LIKE syntax, except (1) Only string types can be compared, (2) The LIKE clause or JSON query must use the same format and text type as data to compare. (3) A single quotation mark does not need to be quoted.
	0, 9, or 17	When the comparison code is 7 (bias +8), the collation field may refer to an ACS, ISR table, ICU collation, or code page name for the LIKE and NOT LIKE operators (see Collation of LIKE Results). When the JSON QUERY operator has been specified, this field should not be provided.

Element	Length (Bytes)	Description
Descriptor (Fixed portion)	2	Number of records to retrieve. To retrieve only one record instead of a set of records, specify 1. In the case of Delete operations, the number of records to delete.
	2	Number of fields to extract from each record. For Delete operations, set to zero.
Descriptor (Repeat this segment for each extracted field)	2	Field length to extract. Not used for Delete operations.
	2	Field offset (zero relative). Not used for Delete operations. **Note:** If field length = 0xFF08 and offset = 0xFFFE (–2), then the 8-byte syskey associated with the record is returned if it has been defined. If field length = 0xFF04 and offset = 0xFFFD (–3), then the 4-byte record length, including variable-length data, is returned.

Collation of LIKE Results

In operations that specify the comparison code as extended operation code 7 (bias +8), the collation field may refer to an existing ACS, ISR table, ICU collation, or code page, as shown in the following table. The format is the signature byte followed by the name.

Type	Total Length (Bytes)	Signature Byte	Name Length (Bytes)	Description
ACS	9	0xAC	8	Unique 8-byte name that identifies the ACS to the MicroKernel Engine. Only existing ACS definitions can be used.
ISR	17	0xAE	16	Unique 16-byte name that identifies the ISR table to the MicroKernel Engine. See *Zen Programmer's Guide* for a list of ISR table names.
ICU	17	0xAE	16	Name a supported ICU collation, either u54-msft_enus_0 or root. You must fill with spaces to 16 bytes.
Code page	17	0xAB	16	Name of a supported code page. You must fill with spaces to 16 bytes.

Using the JSON QUERY Operator

The JSON QUERY operator can be used to filter records that contain data stored as JSON strings. In the record the string data may be stored as ZSTRING or CLOB data types. The filter element must contain the comparison constant as a string that specifies the filter conditions that JSON data in a record must satisfy. This string, also called a JSON query, is itself specified using JSON as shown in the examples given here.

The JSON QUERY operator does not currently support a code page, ACS, or collation. The encoding of the ZSTRING or CLOB data field can be a single-byte Windows ANSI code page or UTF-8.

When you use the JSON QUERY operator, the comparison constant that you specify under the filter term must be a valid JSON query, using JSON syntax as shown in the following example. It must be null-terminated. If the filter string is not in proper JSON format, then the extended operation fails with status code 62 for an incorrect descriptor.

When you use the JSON QUERY operator, if the field of the record to compare against does not follow the JSON specification, the record is treated as failing to meet the filter condition.

JSON Query Examples

The following example shows the syntax required for JSON filtering.

```
query ::= { <expressions> }
<expressions> ::= <json_expression> | <json_expression> , <expressions>
<json_expression> ::=
    "$and" : [ <braced_json_expression> , <braced_json_expressions> ] |
    "$not" : <braced_json_expression> |
    "$or" : [ <braced_json_expression> , <braced_json_expressions> ] |
    <field_expression>
<braced_json_expressions> ::=
    <braced_json_expression> |
    <braced_json_expression> , <braced_json_expressions>
<braced_json_expression> ::= { <json_expression> }
<field_expression> ::=
    <field> : <value> |
    <field> : { "$exists" : false | true } |
    <field> : { "$in" : [ <value> , <values> ] } |
    <field> : { "$nin" : [ <value> , <values> ] } |
    <field> : { "$type" : <type> } |
    <field> : { <operator> : <value> } |
    <field> : { <field_expression> }
<field> ::= <string>
<values> ::= <value> | <value> , <values>
<value> ::= false | true | NULL | <string> | <number> | <array>
<type> ::= "array" | "boolean" | "null" | "number" | "object" | "string"
<operator> ::= "$eq" | "$gt" | "$gte" | "$lt" | "$lte" | "$ne"
```

In a JSON expression, if the query value is not an array and the document value is an array, then each element in the array is evaluated. If any evaluation returns true, then the JSON expression returns true.

In another example, derived from the Bureau of Transportation Statistics data, records of airline flight data in JSON format could have fields resembling the following:

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

To search for records that refer to American Airlines (AA) at Hartsfield-Jackson Atlanta International Airport (ATL), you could use the following query string:

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

Note the use of JSON format for query string. JSON validation tools may be helpful for checking syntax.

Processing of Logical AND and OR in a Filter

The MicroKernel Engine interprets AND and OR operators in a filter with extended operations in strict left-to-right order. It 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 engine accepts this record as meeting the filter condition.
- If the expression is true and the next operator is AND, the engine continues to evaluate each expression until one of the following situations occurs:
    - The engine reaches an OR expression.
    - One of the expressions evaluates to false.
    - The engine reaches the end of the filter.
- If the expression is false and the next operator is OR, the engine continues and evaluates the next expression in the filter.
- If the expression is false and the next operator is AND, the engine rejects the record.

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

- The engine finds the requested number of records that satisfy the filter.
- While the engine 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 engine reaches a rejected record after which no records can satisfy the filter condition in the rest of the file.
- The engine reaches the end of the file.

Examples of Filtering Records

To get the next entire record that satisfies the filter condition, set the filter portion and then 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 filter 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.

Output Buffer for Extended Operations

When you retrieve one or more fields or portions of records with an extended Get or Step operation, you must make sure that the data buffer can hold the information the operation returns. The following table shows the structure of the returned output buffer.

Element	Length (Bytes)	Description
Number of Records	2	For Get and Step operations, number of records returned. For deleted records, number removed.
Repeating portion (one for each record retrieved)		
Length 0	2	Length of first record image retrieved, all fields combined. For Delete operations, it is zero.
Position 0	4 or 8	Physical currency (address) of first record retrieved. For deleted records, address of the first record removed. Size depends on type of entry point: 4 for BTRV or 8 for BTRVEX.
Record 0	*n*	*Image* of first record, all fields combined. Not used for Delete operations.
. . .		(Repeated portions for each record)

Element	Length (Bytes)	Description
Length x	2	Length in bytes of the last record image, all fields combined. For Delete operations, it is zero.
Position x	4 or 8[1]	Physical currency (address) of the last record retrieved or deleted.
Record x	*n*	Image of last record retrieved, all fields combined. Not used for Delete operations.

If all returned records or fields of records are fixed length, your application can simply 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 Engine 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 Engine starts the field description for the next returned field immediately following the data for the current field.

For example, assuming an entry point using 4-byte record addresses, 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 + 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 field of the first record. In such a situation, your application must parse the length, position, and data from the data buffer that the MicroKernel Engine returns.

Result

If the Get Next Extended operation succeeds, the MicroKernel Engine returns the following:

- In the output buffer, one or more fields from one or more records. For details, see Output Buffer for Extended Operations.

- In the output 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 fails, the MicroKernel Engine 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 Engine lost positioning.
134	The MicroKernel Engine cannot read the International Sorting Rule.
135	The specified International Sort Rule table is corrupt or otherwise invalid.
136	The MicroKernel Engine cannot find the specified alternate collating sequence in the file.

The MicroKernel Engine can return a nonzero status code and also valid data, but the last record returned is incomplete. If the buffer length returned is greater than 0, check the buffer for extracted data.

If a field can be only partially filled because the record is too short, then the MicroKernel Engine returns what it can of the record up to and including the partial field. If the partial field is the last field to extract, then the engine continues the operation. Otherwise, it aborts the operation with status code 22.

For example, a Get Next Extended operation retrieves three fields from two variable-length records, the first record 55 bytes long, the second 50 bytes. The output buffer allows 50 bytes for return data. The three 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 Engine performs the Get Next Extended operation, it returns the first record without any problem. However, when it attempts to extract the 10 bytes of field 2, the MicroKernel Engine finds only 5 bytes available, between offset 45 and the end of the record at offset 49. At this point, the MicroKernel Engine does not pad the missing 5 bytes of field 2 and thus cannot extract field 3. Instead, it 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 operators used in the filter condition, the MicroKernel Engine may be able to optimize your request. After reaching a certain rejected record, it returns status code 64, indicating that no records in the rest of the file can satisfy the filter conditions.

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 filter condition and is retrieved, or a record that does not satisfy the filter condition and is rejected, but is still not past the optimization limit. For example, if the extended operation returns status 9 (end of file), the current record is that last record in the file. If status 60 (reject count reached) is returned, then the current record is the last record rejected. If status 64 (filter limit reached) is returned, then the current record is the last one that satisfies the optimization criteria. Even though the MicroKernel Engine had to look at the next record after this to determine that the optimization limit was exceeded, it sets the current record back to the previous record that satisfied the criteria.

Get Position (22)

The Get Position operation (B_GET_POSITION) returns the physical position of the current record. Get Position fails if there is no established physical currency when you issue the operation. Once you determine position (address) of a record, you can use Get Direct/Record (23) to retrieve that record directly by its physical location in the file. The MicroKernel Engine does not perform any disk I/O to process a Get Position request.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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. A minimum of 8 bytes is needed if you are using the BTRVEX entry point.
4. Set the key number to 0.

Result

If the Get Position operation succeeds, the MicroKernel Engine returns the position of the record in the data buffer. The position is a binary value that indicates the offset of the record in the file. The MicroKernel Engine also sets the data buffer length to 4 bytes for BTRV type entry points or 8 bytes for BTRVEX type entry points.

If the Get Position operation fails, the MicroKernel Engine returns one of the following status codes:

3	The file is not open.
8	The current positioning is invalid.
137	The operation with this access method is incompatible. API mismatch. The record address cannot be stored in 4 bytes.

Positioning

The Get Position operation has no effect on positioning.

Get Previous (7)

The Get Previous operation (B_GET_PREVIOUS) 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

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations 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 that established logical position.

Pass the key buffer exactly as the MicroKernel Engine returned it on the previous call, since the information stored there may be needed to determine the current position in the file.

5. Set the key number parameter to the key path used on the previous call that established logical position. You cannot change key paths using a Get Previous operation.

Result

If the Get Previous operation succeeds, the MicroKernel Engine 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 fails, the MicroKernel Engine 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 Engine 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 Delete Extended (86)

The Get Previous Delete Extended operation (B_GET_PREV_EXT_DELETE) 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 compares the examined record or records to a filter condition and retrieves those that match. The filter condition is a logic expression and is not limited to key fields.

As noted under this topic, this operation uses the same input and output buffer structures and returns the result described under Get Next Extended (36). See that operation for more information.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 86.

   By default, the lock bias is no-wait, and any lock bias setting is ignored. Behavior is identical to +500: If the engine cannot delete a locked record, it returns immediately without retrying the operation.

2. Pass the position block for the file.
3. Specify a data buffer to accommodate either the input structure or the returned output, whichever is larger. Initialize the data buffer according to the instructions in Table .
4. Specify the buffer size as either the length of the input structure in Table or the length of the returned output in Table , whichever is larger.

5. Specify the key value from the previous operation in the key buffer that established logical position.

   Pass the key buffer exactly as the MicroKernel Engine returned it on the previous call, since the information stored there may be needed to determine the current position in the file.

6. Set the key number parameter to the key path used on the previous call that established logical position. You cannot change key paths using a Get Previous Delete Extended operation.

Details

The following topics under Get Next Extended (36) cover the structure of the extended operation input buffer and use of its filter segment, as well as the structure of the output buffer that returns the result:

- Input Buffer for Extended Operations
- Output Buffer for Extended Operations

Result

This operation returns the same result as Get Next Delete Extended (85). See that operation for more information.

Positioning

The Get Previous Delete Extended operation does not establish currency. However, you can do a Get Next or Get Previous operation, and the next or previous logical position is valid. A valid current position also becomes available by using Get Position (22) and Get Direct/Record (23).

The following list shows the relationships of selected status codes to filter conditions:

- Status 60 (reject count reached): The current position is a record that does not match the filter condition.
- Status 64 (filter limit reached): The current position is a record that may not match the filter condition. Any attempt to step to the next or previous record will not match the filter condition.
- Status 84 (record or page locked): The current position is a record that may not match the filter condition. It is also possible that the next record matched the filter condition but could not be deleted because of locking.

- Status 22 (data buffer full): The current position is a record that matches the filter condition, but the data buffer does not have space to write the record address, so the MicroKernel Engine did not delete the record.
- Status 9 (end of file): The current position is both logically and physically invalid.

Get Previous Extended (37)

The Get Previous Extended operation (B_GET_PREV_EXTENDED) 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 compares the examined record or records to a filter condition and retrieves those that match. The filter condition is a logic expression and is not limited to key fields.

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

As noted under this topic, this operation uses the same input and output buffer structures and returns the result described under Get Next Extended (36). See that operation for more information.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations Guide*.

2. Pass the position block for the file.
3. Specify a data buffer to accommodate either the input structure or the returned output, whichever is larger. Initialize the data buffer according to the instructions in Table .
4. Specify the buffer size as either the length of the input structure in Table or the length of the returned output in Table , whichever is larger.
5. Specify the key value from the previous operation in the key buffer that established logical position.

   Pass the key buffer exactly as the MicroKernel Engine returned it on the previous call, since the information stored there may be needed to determine the current position in the file.
6. Set the key number parameter to the key path used on the previous call that established logical position. You cannot change key paths using a Get Previous Extended operation.

Details

This operation uses the same input and output buffers as Get Next Extended (36). See that operation for more information.

Result

This operation returns the same result as Get Next Extended (36). See that operation 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 filter condition and is retrieved, or a record that does not satisfy the filter condition and is rejected.

Insert (2)

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

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	Key Buffer	Key Number
Sent	**X**	**X**	**X**	**X**		**X**
Returned		**X**			**X**	

Note: When using the no-currency-change (NCC) option, the Insert 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 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 Engine uses to establish positioning information (currency). To use the NCC option, specify –1 for the key number. To use the system-defined log key (also called system data), specify 125. To use the second system key for system data v2, specify 124.

Result

If the Insert operation succeeds, then the MicroKernel Engine 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 Engine also returns the inserted record in the data buffer, including the autoincrement value assigned by the MicroKernel Engine. An NCC Insert operation does not change the value of the key buffer parameter.

If the Insert operation fails, the MicroKernel Engine 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 logical currency of the application before the NCC Insert.

Note: The MicroKernel Engine 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 Engine 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 (B_EXT_INSERT) inserts one or more records into a file. The MicroKernel Engine adjusts the B-trees for the keys to reflect the key values for the new records.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 .
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 Engine uses to establish currency. To use the NCC option, specify –1 for the key number. To use the system-defined log key (also called system data), specify 125. To use the second system key for system data v2, specify 124.

Details

The following table shows the structure of the input data buffer for Insert Extended operations.

Element	Length (Bytes)	Description
Fixed portion	2 or 4[1]	Number of records inserted.
Repeating portion (one for each record)		
	2 or 4[1]	Length of the record image.
	n	Record image.

[1]Size depends on whether you use BTRV or BTRVEX type entry points.

Result

If the Insert Extended operation succeeds, the MicroKernel Engine places the new records in the file, updates all B-trees to reflect the new records inserted if it is not an 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 2- or 4-byte unsigned integer of the returned data buffer, the MicroKernel Engine places the number of records successfully inserted into the file. Following the count, the MicroKernel Engine stores the record addresses of the inserted records. The following table shows the structure of the output data buffer.

Element	Length (Bytes)	Description
Fixed portion	2 or 4[1]	Number of records inserted.
Repeating portion (one for each record)		
	4 or 8[1]	Record address.

[1]Size depends on whether you use BTRV or BTRVEX type entry points.

If the operation is only partially successful and the MicroKernel Engine returns a nonzero status code, the first 2- or 4-byte unsigned integer 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 Insert Extended is unsuccessful, the MicroKernel Engine 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 key value of the inserted record 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 logical currency of the application prior to the NCC Insert Extended operation.

Note: The MicroKernel Engine 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 Engine 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 (6)operation.

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

Login/Logout (78)

The Login/Logout operation (B_LOGIN/B_LOGOUT) allows a user to specify credentials to obtain authentication and authorization tokens from the database engine. This operation also allows the user to reset the credentials so that they must be entered again to gain access to the database.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	Key Buffer	Key Number
Sent	**X**				**X**	**X**
Returned						

Prerequisites

- The database name and the user ID must be predefined.

Login Procedure

1. Set the operation code to 78.
2. Set the key number to 0.
3. Place the server name, database name, user ID, and password in the key buffer in the form of a database URI. For details about URI connection strings, see Database URIs in *Zen Programmer's Guide*.

Logout Procedure

1. Set the operation code to 78.
2. Set the key number to 1.
3. Place the server name, database name, user ID, and password in the key buffer in the form of a database URI. In *Zen Programmer's Guide*, see Database URIs.

Result

If the Login or Logout operation succeeds, the database engine returns status 0. Otherwise, one of the following status codes may be returned:

1	Invalid operation
172	Database name not found
3103	Unknown server

Notes

The combined length of the database URI must be less than 255 bytes. This is due to the maximum size of the key buffer.

The Login operation has a performance cost. You should not code applications to log in and log out on every file. Instead, log in once to a database at the beginning of a session, then log out when the database work is complete.

Positioning

The Login/Logout operation has no effect on any file currency information.

Open (0)

The Open operation (B_OPEN) 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 path name.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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.

   Place the path name of the file to open in the key buffer parameter. Terminate the path name with a NULL (binary zero) depending on the setting for embedded spaces. The path name can be up to 255 bytes. Any fully-qualified Unified Naming Convention (UNC) path name including the null terminator can be up to 255 bytes long.

   The MicroKernel Engine normally expands the file name to a fully-qualified UNC file name. For example, Z:\Data\File.dat would be converted to \\Servername\ShareName\Data\File.dat. This expanded name must fit into 255 bytes including the null terminator. See also Database URIs in *Zen Programmer's Guide*.

   However, if the Btrieve Open request is sent to a local engine, the MIF will not replace the local drive letter with the computer and share name. Even though a file with a longer path name may be successfully opened locally, remote clients may not be able to open the file.

In client configurations, the Embedded Spaces setting enables support for file names with embedded spaces. By default this setting is on, which means spaces are considered part of the path. When the setting is on, a NULL byte must delimit the file name. When the setting is off, you cannot use file names that contain embedded spaces (such as C:\My Folder\my file.mkd). See Long File Names and Embedded Spaces Support in *Advanced Operations Guide*.

For details about path names supported by Zen clients, see Network Path Formats Supported by Zen Requesters in *Getting Started with Zen*.

4. In the key number parameter, specify one of the mode values listed in Table .

Details

This section describes the open modes that are supported.

Caution! The database engine cannot guarantee transaction atomicity, transaction durability, or archival log safety *for any client* during use of Accelerated mode *by any client*. The reason for this restriction is that in the event a restore from log is needed, the log may not contain adequate information to complete the restore, because it is only a partial record of operations on a data file.

For example, if a system failure occurs while the same file is being accessed by a client performing inserts using Accelerated mode and a client performing updates using Normal mode, it is possible for the transaction log to contain updates to records that do not yet exist in the data files, since the Accelerated insert operation in memory was never flushed to disk, while the transactional update operation was written to the transaction log.

An attempt to roll forward an archival log containing this combination of operations will fail.

When you open a file, you can instruct the MicroKernel Engine through the open modes shown in the following table to use either a local or remote engine. You specify the open mode in the key number parameter.

Note: The Open operation makes no distinction between workstation, workgroup, and server engines when you specify that the local engine should open the file.

Description	No preference	Force local engine	Force remote engine
Normal	0	6	99
Accelerated To improve performance on specific files, you can open a file in Accelerated mode. (The 6.x MicroKernel Engine accepted accelerated mode opens, but interpreted them as normal opens.) When you open a file in Accelerated mode, the MicroKernel Engine does not perform transaction logging on the file. See the caution above.	–1	7	100
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 Engine cannot automatically recover. If data in the file indexes has been corrupted, you can retrieve the records by opening the file in read-only mode and then using Step Next (24).	–2	8	101
Writable If the MicroKernel Engine cannot open a file for writing, then the writable mode returns an error. One common scenario is a file without file system write permission.	–3	9	102
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.	–4	10	103

There is no fixed limit on the maximum number of open files. The number of files that can be opened at once depends on the available memory.

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

Result

If the Open operation succeeds, the MicroKernel Engine 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 fails, the MicroKernel Engine returns one of the following status codes:

2 The application encountered an I/O error.
11 The specified file name is invalid.
12 The MicroKernel Engine cannot find the specified file.
20 The MicroKernel Engine 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 table shows open mode combinations involving local clients.

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

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

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 (B_RESET) 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

	Op Code	Pos Block	Data Buf	Data Buf Len	Key Buffer	Key Number
Sent	**X**				**X**	**X**
Returned						

Prerequisites

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

Procedure

1. Set the operation code to 28.
2. Set the key number and key buffer to 0.

Result

If the Reset operation succeeds, the MicroKernel Engine 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 fails, the MicroKernel Engine 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 (B_SET_DIR) sets the current directory to a specified path name.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 Engine uses the default drive. If you do not specify the complete path for the directory, the MicroKernel Engine appends the directory path specified in the key buffer to the current directory.

   For details about path names supported by Zen clients, see Network Path Formats Supported by Zen Requesters in *Getting Started with Zen*.

Result

If the Set Directory operation succeeds, the MicroKernel Engine makes the directory specified in the key buffer the current directory.

If the Set Directory operation fails, the MicroKernel Engine 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 (B_SET_OWNER) assigns an owner name to a file, which serves as an access password. If a file is given an owner name, users or applications must provide that string each time they access the file. You can specify that an owner name be required for any access or only for update privileges. Owner names are in ASCII and in the case of long owner names can also be hexadecimal. When you assign an owner name, you can also direct the MicroKernel Engine to encrypt the file data on the disk. If so, the MicroKernel Engine encrypts all data during the Set Owner operation. Performance may be affected, since longer files lengthen Set Owner execution time. For more information, see Owner Names in *Advanced Operations Guide*.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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.

   Optionally, you can include a bias of +17000 to create a long owner name up to 24 bytes in length. This bias is also defined in btrconst.h as B_LONG_OWNER_NAME_BIAS.

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 Engine requires that the name be in both buffers to avoid accidentally providing an incorrect value.

   If the +17000 bias is not set, a short owner name can be up to 8 bytes long and must end with a binary 0. If the +17000 bias is set, a long owner name can be used and must end with a binary 0. In either case, the owner name cannot consist of all spaces (0x20). The length of a long owner name depends on the file format. For more information, see Owner Names.

4. Set the length of the owner name, including the binary 0, in the data buffer length parameter.
5. Set the key number to an integer that specifies the type of access restrictions and encryption for the file. The following table lists the four key numbers and their results.

Key Number	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).

Details

Once you specify an owner name, it remains in effect until you issue a Clear Owner (30) operation. The table above lists the access restriction codes you can set for the key number.

Result

If the Set Owner operation succeeds, the MicroKernel Engine 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 succeeds, the MicroKernel Engine encrypts the data in the file if encryption is specified. Encryption begins immediately. The MicroKernel Engine has control until the entire file is encrypted.

Regarding performance, it is helpful to note that the MicroKernel Engine decrypts an encrypted page when it loads it from the disk and encrypts it before writing it to the disk again. Reading data from an encrypted file is slower than reading data from an unencrypted file. Also, the larger the file, the longer it takes to encrypt or decrypt. In an encrypted file scenario, if you have a small cache or use a relatively large number of modification operations, the MicroKernel Engine must execute the encryption routine more frequently.

If the Set Owner operation fails, the MicroKernel Engine returns one of these status codes:

41 The MicroKernel Engine does not allow the attempted operation.

50 An owner name is already set.

51 The submitted owner name is invalid.

Positioning

The Set Owner operation has no effect on positioning.

Stat (15)

The Stat operation (B_STAT) uses the data buffer to retrieve statistics about the file specification, such as the number of records it contains, the number of unique key values stored for each of its indexes, its number of empty pages, and any alternate collating sequences (ACS). New keys and ACS values may have been added since the file was created. Since you must account for this new information, you may not be able to reuse the original data buffer size that was used for the Create (14) operation.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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.
5. Specify a key buffer at least 255 characters long.
6. Set the key number as follows:
   - 0 to exclude file version
   - –1 to include file version

Details

Because Create (14) and Stat (15) use the same data buffer structure, they are documented together under Create (14) with the slight differences noted.

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 0x95 indicates that the file is a 9.5 file, a value of 0x80 indicates that the file is 8.x, and so on. When creating a file, the MicroKernel Engine assigns a version number according to these attributes.
- The Number of Records is a 4- or 8-byte value representing the number of records in the file.
- In the File Flags word, Bits 9(0x0200) and 12 (0x1000) have the following meaning:

Bit 9 = 1 and Bit 12 = 0	File was created with system data or system data v2. This does not necessarily mean that the system keys are currently in use. It may have been dropped. See Stat Extended (65).
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.

- Number of Unused Duplicate Pointers indicates how many unused duplicate pointers remain in the file.
- The reserved areas are allocated even though the MicroKernel Engine ignores them on a Stat operation.

Key Specifications

The Key Specification fields in the returned data buffer are the same as those described in Table , except that a 4- or 8-byte Number of Unique Key Values indicates the number of records that have a unique, nonduplicate value for the specified key.

Alternate Collating Sequences

The alternate collating sequences (ACS) in the returned data buffer are the same as those described under Create (14).

Result

If the Stat operation succeeds, the MicroKernel Engine 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 Engine returns the file name of the first extension file in the key buffer. If the file name of the first extension file is longer than 63 bytes, the MicroKernel Engine truncates the file name. If the file is not an extended file, the MicroKernel Engine initializes the first byte of the key buffer to 0. You can also use the Stat Extended (65) operation to retrieve information on extended files.

If the Stat operation fails, the MicroKernel Engine 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 (B_EXTENDED_STAT) has several subfunctions that allow an application to gather information about an open file.

Subfunction ID	Description
1	Listing of extension file names
2	System data information for the file
3	Duplicate conflict record and key identification
4	File information
5	Gateway identification
6	Lock owner identification
7	Security information
8	Listing of table or file name causing a status code 71 (a violation of the referential integrity definitions)

See the following subfunction topics for more information.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	Key Buffer	Key Number
Sent	**X**	**X**	**X**	**X**		**X**
Returned		**X**	**X**	**X**		

Prerequisites

The file must be open.

Procedure

1. Set the operation code to 65.
2. Pass the position block for the given file.

3. Store the stat extended structure in the data buffer. See the subsections below for more information about the stat extended structure required for each subfunction.
4. Specify the data buffer length.
5. Set the key number to 0.

Subfunction 1: Extended File Information

For the file specified by the input position block, this subfunction returns information about the extension files associated with the specified data file. Returned information includes number of extension files that exist, number returned by the function, and file names for the returned files.

Input Data Buffer Structure

To receive information about extension files, you must create an extended files descriptor in the data buffer, as shown in the following table.

Element	Length (Bytes)	Description
Signature	4	Type of stat extended call. Specify the following 4 bytes to indicate a stat extended call: 0x45, 0x78, 0x53, 0x74. These are equivalent to ASCII *ExSt* or to the value 0x74537845 on Intel-types, LoHi and little-endian hardware.
Subfunction	4	Type of stat extended call. Specify 0x00000001.
Namespace	4	File naming convention. Specify 0x00000000.
Max Files	4	Maximum number of file names 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 file name 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 Engine returns status code 0 and no file names.
Buffer space	*n*	Allow enough additional space for the return data. If you receive status code 22, retry the operation with a larger data buffer size.

Output Data Buffer Structure

For the extended files subfunction, the MicroKernel Engine updates the value of the data buffer length parameter and returns an extended files structure in the data buffer, as illustrated in the following table.

Element	Length (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.
File Name Portion (Repeated for each file name returned)		
Length of File Name	4	Length of the extension file name.
File Name	*n*	Extension file name.

Subfunction 2: System Data Information

For the file specified by the input position block, this subfunction returns information about whether a file has system keys defined, and whether the file can be logged (transaction durable).

Input Data Buffer Structure

To receive information about use of system data in a file, you must create a system data descriptor in the data buffer, as shown in the following table.

Element	Length (Bytes)	Description
Signature	4	Type of stat extended call. Specify the following 4 bytes to indicate a stat extended call: 0x45, 0x78, 0x53, 0x74. These are equivalent to ASCII *ExSt* or to the value 0x74537845 on Intel-types, LoHi and little-endian hardware.
Subfunction	4	Type of stat extended call. Specify 0x00000002.

Output Data Buffer Structure

For the system data subfunction, the MicroKernel Engine returns a system data structure in the data buffer, as follows.

Element	Length (Bytes)	Description
Has System Data	1	Indicates whether the file has system data defined. 0 = None, 1 = Has System Data, 2 = Has System Data v2.
Has System Keys	1	Indicates presence of system keys. 0 = None, 1 = System key 125, 2 = System key 124, 3 = Both.
Is Loggable	1	Indicates whether the file has any unique key that can be used for transaction logging and durability. The key can be user- or system-defined. 1 = Yes and 0 = No.
Log Key Number	1	Key number used as a transaction log key. If the system-defined key is the one being used, this value is 125.
Size of System Data	2	8 = System data only in key 125. 16 = System data v2 in keys 125 and 124.
Engine Major Version	2	A two-byte field that contains the major version of the database engine.

Subfunction 3: Duplicate Record Conflict Information

For the file specified by the input position block, this subfunction returns information about duplicate record conflicts. Returned information includes the record address and key number that caused a status code 5 (Duplicate Key) on a previous failed insert or update operation.

Input Data Buffer Structure

To receive information about the record address and key number that caused the most recent status code 5 (duplicate key) to report duplicate record conflicts, you must create a duplicate record information descriptor in the data buffer, as follows.

Element	Length (Bytes)	Description
Signature	4	Type of stat extended call. Specify the following 4 bytes to indicate a stat extended call: 0x45, 0x78, 0x53, 0x74. These are equivalent to ASCII *ExSt* or to the value 0x74537845 on Intel-types, LoHi and little-endian hardware.

Element	Length (Bytes)	Description
Subfunction	4	Type of stat extended call. Specify 0x00000003.

Output Data Buffer Structure

For the duplicate record conflict subfunction, the MicroKernel Engine returns a duplicate record conflict structure in the data buffer, as follows.

Element	Length (Bytes)	Description
Duplicate Record Address	4 or 8[1]	Physical address of record containing the duplicate key value. The 13.0 file format requires 8 bytes.
Key Number	2	Key number of the key containing the duplicate value.

[1]Size depends on whether you use BTRV or BTRVEX type entry points.

Subfunction 4: File Information

For the file specified by the input position block, this subfunction returns file information. Returned information includes: the internal file ID used by the MicroKernel Engine to identify the file, the number of file handles currently open, the timestamp of the last time the file was opened, and a variety of flags indicating file properties.

Input Data Buffer Structure

To receive information about an open file, you must create a file information descriptor in the data buffer, as follows.

Element	Length (Bytes)	Description
Signature	4	Type of stat extended call. Specify the following 4 bytes to indicate a stat extended call: 0x45, 0x78, 0x53, 0x74. These are equivalent to ASCII *ExSt* or to the value 0x74537845 on Intel-types, LoHi and little-endian hardware.
Subfunction	4	Type of stat extended call. Specify 0x00000004.

Element	Length (Bytes)	Description
Buffer space	12	Additional bytes needed for returned information. See Output Data Buffer Structure. For the file information subfunction, the MicroKernel Engine returns a file information structure in the data buffer.

Output Data Buffer Structure

For the file information subfunction, the MicroKernel Engine returns a file information structure in the data buffer, as follows.

Element	Length (Bytes)	Description
FileID	4	A unique number which the MicroKernel Engine uses to identify the file.
Number of Handles	4	The current number of handles that the MicroKernel Engine has open on this file.
Open Time Stamp	4	The system time when the physical file was last opened by the MicroKernel Engine. This system time is expressed as the number of seconds since midnight on January 1, 1970 in coordinated universal time (UTC) time.
File Usage Count	4	This number increments at each checkpoint or System Transaction. It is also the usage count placed in the FCR. The number returned here is the usage count of the file as it is represented in the MicroKernel Engine cache. When a checkpoint starts, this number increments.
Flags	4	A four-byte bitmap in which various values may be set. See the following table for descriptions of the possible values. More flags may be added in the future.

The permitted values for the Flags field are described in the following tables.

Value	Name	Description
0x00000001	Explicit Locks	There are explicit locks currently on the file.
0x00000002	Client Transactions	There is at least one client transaction currently open on the file.
0x00000004	Read Only	The file was opened by the MicroKernel Engine as Read Only. This may be a CD-ROM drive or a read-only directory.

Value	Name	Description
0x00000008	Continuous Operations	The file is currently in continuous operations.
0x00000010	Referential Integrity	The file has referential integrity constraints on it.
0x00000020	Owner Read/Write	The file has a Read/Write Owner name assigned to it. The owner name is required to read from or write to the file.
0x00000040	Owner Reads OK	The file has an owner name that is required only to write to the file. Reads can be done without an owner name.
0x00000080	Opened with Wrong Owner	The file has a Reads-OK Owner name assigned to it and the handle was opened with the wrong owner name.
0x00000100	Owner Encryption	The file has the encryption flag on the owner name. This flag means that every page in the file is encrypted and cannot be read using a text editor.
0x00000200	Opened by Cache Engine	If set, this file has been opened by a cache engine.
0x00000400	Traditional Encryption	The file is encrypted using the Btrieve traditional encryption algorithm.
0x00000800	128-Byte Encryption	The file is encrypted using a 128-byte encryption algorithm.
0x00001000	AES-192 Encryption	The file is encrypted using AES-192 encryption.
0x00002000	AES-256 Encryption	The file is encrypted using AES-256 encryption.

Subfunction 5: Gateway Information

For the file specified by the input position block, this subfunction returns information about the Gateway engine that has control of the file.

Input Data Buffer Structure

To receive information about the Gateway engine that is responsible for the specified file, you must create a gateway information descriptor in the data buffer, as follows.

Element	Length (Bytes)	Description
Signature	2	Type of stat extended call. Specify the following 4 bytes to indicate a stat extended call: 0x45, 0x78, 0x53, 0x74. These are equivalent to ASCII *ExSt* or to the value 0x74537845 on Intel-types, LoHi and little-endian hardware.
Subfunction	2	Type of stat extended call. Specify 0x00000005.
Buffer Space	at least 80	Additional bytes needed for returned information. See Output Data Buffer Structure for details.

Output Data Buffer Structure

For the gateway information subfunction, the MicroKernel Engine modifies the data buffer length parameter and returns a file information structure in the data buffer, as follows.

Element	Length (Bytes)	Description
Major Version	2	The major version of the engine, such as version 7 or 8.
Minor Version	2	The minor version of the engine, such as 05 or 82.
Patch Level	2	The patch level of the engine, such as 1, 2, or 3.
Platform	2	The type of requester or engine, as listed under Version (26).
Server Name	64	A null-terminated string indicating the name of the machine where the database engine is running. The data buffer length returned by the Btrieve API call contains the actual length of the data returned, including the server name and the null terminator.

Subfunction 6: Lock Owner Identification

For the file specified by the input position block, this subfunction returns information about the cause of the most recent status code 84 or 85 that occurred when accessing the file.

Input Data Buffer Structure

To receive information about the cause of a status code 84 or 85, you must create a lock owner information descriptor in the data buffer, as follows.

Element	Length (Bytes)	Description
Signature	4	Unique identifier for a stat extended call. Specify *ExSt*. See Subfunction 1: Extended File Information.
Subfunction	4	Type of stat extended call. Specify 0x00000006.
Buffer Space	at least 96	Additional bytes needed for returned information. See Subfunction 7: Security Information for details.

Output Data Buffer Structure

For the lock owner information subfunction, the MicroKernel Engine modifies the data buffer length parameter and returns a file information structure in the data buffer, as follows.

Element	Length (Bytes)	Description
Client ID	16	The 16-byte client ID of the blocking client.
Flags	4	A four-byte bitmap containing flags indicating the type of conflict that occurred. See the following table for a description of each flag value.
Time In Transaction	4	Number of milliseconds in which the blocking client has been in a transaction. This can be helpful in determining whether to retry the operation.
Key Number	4	If the conflict occurred on a key page, this element indicates which key is involved. Tracking this information can be useful in designing a database with fewer potential conflicts.
Transaction Level	4	If this number is nonzero, then the blocking client is currently in a transaction. Since some page and record locks are held until the transaction completes, this information might be useful in determining if the operation should be retried.
Reserved	8	Reserved for future use. If there is some information about the blocking client which you think may be useful, please contact technical support.

Element	Length (Bytes)	Description
Display Name	64	This is a null-terminated string which is the same identifying name that is displayed in Monitor for each client. Use at least 64 bytes since that is the current maximum display name length. The data buffer length returned by the Btrieve API call contains the actual length of the data returned, including the display name and the null terminator.

If there is no record in the MicroKernel Engine of a previous blocking client, then the output data buffer length is set to zero.

The permitted values for the Flags field are described in the following table.

Value	Name	Description
0x00000001	Implicit Lock	The blocking client is using an implicit lock.
0x00000002	Explicit Lock	The blocking client is using an explicit lock.
0x00000010	File Lock	The blocking client is using a file lock.
0x00000020	Page Lock	The blocking client is using a page lock.
0x00000040	Record Lock	The blocking client is using a record lock.
0x00000100	Data Page	If the conflict was a Page Lock, this flag indicates the conflict occurred on a data page.
0x00000200	Key Page	If the conflict was a Page Lock, this flag indicates the conflict occurred on a key page.
0x00000400	Variable Page	If the conflict was a Page Lock, this flag indicates the conflict occurred on a variable page.
0x00000800	Same Process	If this flag is set, then the first 12 bytes of the blocking client ID are the same as the first 12 bytes of the client that got blocked, that is, the client that is issuing the Stat Extended call. In this case, it means that the two blocking clients came from the same process on the same system. If you have a single threaded application making Btrieve API calls, then retrying this operation will not help. You need to complete or abort the work that is blocking.
0x00001000	Write No Wait	Indicates that the blocking client is using the 500 bias.

Value	Name	Description
0x00002000	Write Hold	Indicates that the blocking client made a change to a page that caused that client to keep the full page lock until its transaction completes. This situation can occur on implicit key page locks when a change causes key entries to move to another page.
0x00004000	Read No Wait	For explicit record locks, this flag indicates that the blocking client is using either lock bias 200 or 400.
0x00008000	Read Multiple	For explicit record locks, this flag indicates that the blocking client is using either lock bias 300 or 400.

Subfunction 7: Security Information

This subfunction returns information about how the client was authenticated and authorized to access the current file. It also shows information about the current database being used for security.

Input Data Buffer Structure

To receive security information about how this handle is authenticated and what permissions it has, you must create a security information descriptor in the data buffer, as follows.

Element	Length (Bytes)	Description
Signature	4	Type of stat extended call. Specify the following 4 bytes to indicate a stat extended call: 0x45, 0x78, 0x53, 0x74. These are equivalent to ASCII *ExSt* or to the value 0x74537845 on Intel-types, LoHi and little-endian hardware.
Subfunction	4	Type of stat extended call. Specify 0x00000007.
Buffer Space	at least 142	Additional bytes needed for returned information. See Result for details.

Output Data Buffer Structure

For the security information subfunction, the MicroKernel modifies the data buffer length parameter and returns a file information structure in the data buffer, as follows.

Element	Length (Bytes)	Description
Flags for Handle	4	A four-byte bitmap containing flags indicating the methods used for security on this handle. See the following table for a description of each flag value.
Flags for Database	4	A four-byte bitmap containing flags indicating the methods used for security on the current default database for this client. See the following table for a description of each flag value.
Permissions	4	These are the permissions granted to the client using this handle. See the permission following table for a description of each flag value.
Buffer Size for Handle Database Name	2	Length of the buffer used to store the null-terminated Handle Database Name string.
Buffer Size for Handle Table Name	2	Length of the buffer used to store the null-terminated Handle Table Name string. **Note:** The table name will not be known unless the file is bound to a database (Referential Constraints, for example), or the file was opened using a URI connection string that referred to the file by its Table Name. For details about URI connection strings, see Database URIs in *Zen Programmer's Guide*.
Buffer Size for Handle User Name	2	Length of the buffer used to store the null-terminated Handle User Name string.
Buffer Size for Current Database Name	2	Length of the buffer used to store the null-terminated Current Database Name string.
Buffer Size for Current User Name	2	Length of the buffer used to store the null-terminated Current User Name string.
Handle Database Name	Variable	The database name used to establish security for this handle.
Handle Table Name	Variable	The table name associated with this handle
Handle User Name	Variable	The user name used to establish security for this handle.

Element	Length (Bytes)	Description
Current Database Name	Variable	The current default database name for this client.
Current User Name	Variable	The user name associated with the current default database for this client.

The permitted values for the two Flags fields are described in the following tables.

Value	Name	Security Flag Description
0x00000001	Trusted	This handle is trusted, no database is assigned.
0x00000002	Implicit	Database login is implicit - during the open.
0x00000004	Explicit	Database login is explicit - a Btrieve login was made.
0x00000008	Authentication by Database	Authentication was done by database security. If not set, authentication was done using operating system security.
0x00000010	Authorization by Database	Authorization was done by database security. If not set, authorization was done using operating system security.
0x00000020	Windows Named Pipe	If authentication is by the operating system, this indicates that an NT named pipe connection was used for security.
0x00000040	Workgroup	If authentication is by the operating system, this indicates that Workgroup Engine style security was done, which means that no authentication or authorization was done.
0x00000080	Btpasswd	If authentication is by a Linux, macOS, or Raspbian operating system, this indicates that etc/btpasswd file was used.
0x00000100	PAM	If authentication is by the Linux, macOS, or Raspbian operating system, this indicates that PAM authentication was used.
0x00000200	RTSS Complete	If authentication is by the operating system, this indicates that authentication was done using RTSS with the Complete setting.
0x00000400	RTSS Preauthorized	If authentication is by the operating system, this indicates that authentication was done using RTSS with the Preauthorized setting.
0x00000800	RTSS Disabled	If authentication is by the operating system, this indicates that authentication was done using RTSS with the Disabled setting.

Value	Name	Permission Flag Description
0x00000000	No Rights	No rights to the database object. No permission granted.
0x00000001	Open	Permission granted to open the file. This also implies that the records can be read.
0x00000002	Insert	Permission granted to insert records.
0x00000004	Update	Permission granted to update records.
0x00000008	Create	Permission granted to create this file.
0x00000010	Delete	Permission granted to delete records.
0x00000020	Execute	Permission granted to execute stored procedures in SQL.
0x00000040	Alter	Permission granted to alter this file in SQL.
0x00000080	Refer	Permission granted to refer to this file in SQL.
0x00000100	Create View	Permission granted to create views to this file in SQL.
0x00000200	Create Stored Procedure	Permission granted to create stored procedures for this file in SQL.

Subfunction 8: Listing of Table or File Name Causing a Status Code 71

This subfunction returns information on the table or data file that caused a status code 71, a violation of the referential integrity definitions. Returned information includes the file name, Btrieve operation code, and position of the record that caused the referential integrity error.

Input Data Buffer Structure

To receive information about an open file, you must create a file information descriptor in the data buffer, as follows.

Element	Length (Bytes)	Description
Signature	4	Type of stat extended call. Specify the following 4 bytes to indicate a stat extended call: 0x45, 0x78, 0x53, 0x74. These are equivalent to ASCII *ExSt* or to the value 0x74537845 on Intel-types, LoHi and little-endian hardware.

Element	Length (Bytes)	Description
Subfunction	4	Type of stat extended call. Specify 0x00000008.

Output Data Buffer Structure

For the file information subfunction, the MicroKernel Engine returns a file information structure in the data buffer, as follows. The data buffer supplied must be large enough to hold the data returned.

Element	Length (Bytes)	Description
File Name	255	File name causing the RI error.
Btrieve Op Code	4	Btrieve operation code that caused the RI error.
Record Position	4 or 8[1]	Physical record position of the record that caused the RI error. The 13.0 file format requires 8 bytes.

[1]Size depends on whether you use BTRV or BTRVEX type entry points.

Result

If the Stat Extended operation fails, the MicroKernel Engine 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 (B_STEP_FIRST) retrieves the first physical record of the file. The MicroKernel Engine does not use a key path to retrieve the record.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations 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 succeeds, the MicroKernel Engine returns the first physical record of the file in the data buffer and sets the data buffer length parameter to the number of bytes returned.

If the Step First operation fails, the MicroKernel Engine 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 (B_STEP_LAST) retrieves the last physical record of the file. The MicroKernel Engine does not use a key path to retrieve the record.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations 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 succeeds, the MicroKernel Engine returns the last physical record of the file in the data buffer and sets the data buffer length parameter to the number of bytes returned.

If the Step Last operation fails, the MicroKernel Engine 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 (B_STEP_NEXT) retrieves the record to which the next physical position points. The MicroKernel Engine 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

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations 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 succeeds, the MicroKernel Engine returns the next physical record in the file in the data buffer and sets the data buffer length parameter to the number of bytes returned.

If the Step Next operation fails, the MicroKernel Engine 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 (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 (0), it returns the first record in the file.

Step Next Extended (38)

The Step Next Extended operation (B_STEP_NEXT_EXT) examines one or more records, starting at the next physical position and proceeding toward the end of the file. It compares the examined record or records to a filter condition and retrieves those that match. The filter condition is a logic expression and is not limited to key fields.

Step Next Extended can also extract specified fields from existing records and return a new set of records that contain only the extracted fields.

As noted under this topic, this operation uses the same input and output buffer structures and returns the result described under Get Next Extended (36). See that operation for more information.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations Guide*.
2. Pass the position block for the file.

3. Specify a data buffer to accommodate either the input structure or the returned output, whichever is larger. Initialize the data buffer according to the instructions in Table .
4. Specify the buffer size as either the length of the input structure in Table or the length of the returned output in Table , whichever is larger.
5. Specify the data buffer length from the preceding step.

Details

The Step Next Extended operation shares the same details as the Get Next Extended operation. See the Details topic for that operation for more information.

Result

If the Step Next Extended operation succeeds, the MicroKernel Engine returns one or more fields from one or more records to the data buffer (as shown in Table). The MicroKernel Engine also sets the data buffer length parameter to the number of bytes it returned to the data buffer.

If the Step Next Extended operation fails, the MicroKernel Engine 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.
65	The field offset is incorrect.
82	The MicroKernel Engine lost positioning.
134	The MicroKernel Engine cannot read the International Sorting Rule.
135	The specified International Sort Rule table is corrupt or otherwise invalid.
136	The MicroKernel Engine cannot find the specified alternate collating sequence in the file.

The MicroKernel Engine can return a nonzero status code and also valid data, but the last record returned is incomplete. If the buffer length returned is greater than zero, check the output buffer for extracted data.

If a field can be only partially filled because the record is too short, then the MicroKernel Engine returns what it can of the record up to and including the partial field. If the partial field is the last

field to extract, then the engine continues the operation. Otherwise, it aborts the operation with status code 22.

For example, a Step Next Extended operation retrieves three fields from two variable-length records, the first record 55 bytes long, the second 50 bytes. The output buffer allows 50 bytes for return data. The three 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 Engine 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 Engine finds that only 5 bytes are available (between offset 45 and the end of the record, at offset 49). At this point, the MicroKernel Engine does not pad the missing 5 bytes of field 2, and thus cannot extract field 3. Instead, the MicroKernel Engine returns status code 22 and places all of field 1 and first 5 bytes of field 2 in the return data buffer.

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

Positioning

The Step Next Extended operation does not establish any logical currency, but the last record examined becomes the current physical record. This record can be either a record that satisfies the filter condition and is retrieved or a record that does not satisfy the filter condition and is rejected.

Step Next Delete Extended (87)

The Step Next Delete Extended operation (B_STEP_NEXT_EXT_DELETE) examines one or more records, starting at the next physical position and proceeding toward the end of the file. It compares the examined record or records to a filter condition and deletes those that match. The filter condition is a logic expression and is not limited to key fields.

As noted under this topic, this operation uses the same input and output buffer structures and returns the result described under Get Next Extended (36). See that operation for more information.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 87.

   By default, the lock bias is no-wait, and any lock bias setting is ignored. Behavior is identical to +500: If the engine cannot delete a locked record, it returns immediately without retrying the operation.

2. Pass the position block for the file.
3. Specify a data buffer to accommodate either the input structure or the returned output, whichever is larger. Initialize the data buffer according to the instructions in Table .
4. Specify the buffer size as either the length of the input structure in Table or the length of the returned output in Table , whichever is larger.
5. Specify the data buffer length from the preceding step.

Details

The following topics under Get Next Extended (36) cover the structure of the extended operation input buffer and use of its filter segment, as well as the structure of the output buffer that returns the result:

- Input Buffer for Extended Operations
- Output Buffer for Extended Operations

Result

If the Step Next Delete Extended operation succeeds, the MicroKernel Engine deletes one or more records. The MicroKernel Engine also sets the data buffer length parameter to the number of bytes it returned to the data buffer.

If the Step Next Delete Extended operation fails, the MicroKernel Engine 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.
65	The field offset is incorrect.
82	The MicroKernel Engine lost positioning.
134	The MicroKernel Engine cannot read the International Sorting Rule.
135	The specified International Sort Rule table is corrupt or otherwise invalid.
136	The MicroKernel Engine cannot find the specified alternate collating sequence in the file.

If the output buffer length is zero, then no records were deleted. However, the operation may have succeeded in deleting some records before failing. The following list gives some examples of these partial successes:

- The output buffer no longer has room to write out the record address for the current record that matches the filter conditions. That record is not deleted, and the operation fails with status code 22.
- Another client has locked the current record, and the operation fails with status code 84.

In these cases, the output buffer length is greater than zero, and the first two bytes of the buffer give a count of the number of deleted records.

Positioning

The Step Next Delete Extended operation does not establish currency. When a record is deleted, both the logical and physical current positions are no longer valid. However, you can do a Step Next or Step Previous operation because those physical positions are accessible and then have a valid position. If the deleted record has been reached by a Get operation, the next and previous logical positions are also valid. A valid current position becomes available when those operations are called or by using Get Position (22) and Get Direct/Record (23).

The following list shows the relationships of selected status codes to filter conditions:

- Status 60 (reject count reached): The current position is a record that does not match the filter condition.
- Status 84 (record or page locked): The current position is a record that may not match the filter condition. It is also possible that the next record matched the filter condition but could not be deleted because of locking.
- Status 22 (data buffer full): The current position is a record that matches the filter condition, but the data buffer does not have space to write the record address, so the MicroKernel Engine did not delete the record.
- Status 9 (end of file): The current position is both logically and physically invalid.

Step Previous (35)

The Step Previous operation (B_STEP_PREVIOUS) retrieves the record to which the previous physical position points. The MicroKernel Engine 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

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations 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 succeeds, the MicroKernel Engine 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 fails, the MicroKernel Engine 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 Delete Extended (88)

The Step Previous Delete Extended operation (B_STEP_PREV_EXT_DELETE) examines one or more records, starting at the previous physical position and proceeding toward the beginning of the file. It compares the examined record or records to a filter condition and deletes those that match. The filter condition is a logic expression and is not limited to key fields.

As noted under this topic, this operation uses the same input and output buffer structures and returns the result described under Get Next Extended (36). See that operation for more information.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 88.

   By default, the lock bias is no-wait, and any lock bias setting is ignored. Behavior is identical to +500: If the engine cannot delete a locked record, it returns immediately without retrying the operation.

2. Pass the position block for the file.
3. Specify a data buffer to accommodate either the input structure or the returned output, whichever is larger. Initialize the data buffer according to the instructions in Table .
4. Specify the data buffer length from the preceding step.

Details

The following topics under Get Next Extended (36) cover the structure of the extended operation input buffer and use of its filter segment, as well as the structure of the output buffer that returns the result:

- Input Buffer for Extended Operations
- Output Buffer for Extended Operations

Result

This operation returns the same result as Step Next Delete Extended (87). See that operation for more information.

Positioning

The Step Previous Delete Extended operation does not establish currency. When a record is deleted, both the logical and physical current positions are no longer valid. However, you can do a Step Next or Step Previous operation because those physical positions are accessible and then have a valid position. If the deleted record has been reached by a Get operation, the next and previous logical positions are also valid. A valid current position becomes available when those operations are called or by using Get Position (22) and Get Direct/Record (23).

The following list shows the relationships of selected status codes to filter conditions:

- Status 60 (reject count reached): The current position is a record that does not match the filter condition.
- Status 84 (record or page locked): The current position is a record that may not match the filter condition. It is also possible that the next record matched the filter condition but could not be deleted because of locking.
- Status 22 (data buffer full): The current position is a record that matches the filter condition, but the data buffer does not have space to write the record address, so the MicroKernel Engine did not delete the record.
- Status 9 (end of file): The current position is both logically and physically invalid.

Step Previous Extended (39)

The Step Previous Extended operation (B_STEP_PREVIOUS_EXT) examines one or more records, starting at the previous physical position and proceeding toward the beginning of the file. It compares the examined record or records to a filter condition and retrieves those that match. The filter condition is a logic expression and is not limited to key fields.

Step Previous Extended can also extract specified fields from existing records and return a new set of records that contain only the extracted fields.

As noted under this topic, this operation uses the same input and output buffer structures and returns the result described under Get Next Extended (36). See that operation for more information.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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 details about record locking and data integrity, see *Zen Programmer's Guide*, as well as the Wait Lock Timeout property for configuring Zen servers in *Advanced Operations Guide*.
2. Pass the position block for the file.

3. Specify a data buffer to accommodate either the input structure or the returned output, whichever is larger. Initialize the data buffer according to the instructions in Table .
4. Specify the data buffer length from the preceding step.

Details

This operation uses the same input and output buffers as Get Next Extended (36). See that operation for more information.

Result

This operation returns the same result as Get Next Extended (36). See that operation for more information.

Positioning

The Step Previous Extended operation does not establish logical currency, but the last record examined becomes the current physical record. This record can be either a record that satisfies the filter condition and is retrieved or a record that does not satisfy the filter condition and is rejected.

Stop (25)

The Stop operation (B_STOP) performs a number of termination routines for the client, such as releasing all locks and closing all open files associated with that client.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	Key Buffer	Key Number
Sent	**X**					
Returned						

Procedure

Set the operation code to 25.

Result

If the Stop operation succeeds, the MicroKernel Engine 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 Engine) exist and depending on the MicroKernel Engine configuration, the MicroKernel Engine may terminate itself and free a number of resources.

If the Stop operation fails, the MicroKernel Engine returns a nonzero status code. The most common nonzero status code is 20 (Record Manager Inactive). This status occurs because the MicroKernel Engine or the Requester is not loaded.

Positioning

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

Unlock (27)

The Unlock operation (B_UNLOCK) 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

	Op Code	Pos Block	Data Buf	Data Buf Len	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

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 nonnegative value.

To unlock a record locked by a multiple-record lock

1. Retrieve the 4- or 8-byte position of the record to unlock by issuing a Get Position (22) for that record. Then, continue to the next steps to issue the Unlock operation.
2. Set the operation code to 27.
3. Pass the position block for the file that contains the locked record.

4. Store in the data buffer the 4- or 8-byte position that Get Position (22) returned. Use the same BTRV or BTRVEX type entry point for Get Position and Unlock to be consistent with record address size.
5. Set the data buffer length to 4 or 8.
6. Set the key number parameter to –1.

To unlock all the multiple record locks on a file

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 succeeds, the MicroKernel Engine releases all the locks that the operation specified.

If the Unlock operation fails, the MicroKernel Engine returns a nonzero status code – most likely, status code 81.

Positioning

The Unlock operation has no effect on positioning.

Update (3)

The Update operation (B_UPDATE) changes the information in an existing record.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	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. Note that although an Extended Get, Extended Step, or Get Key operation establishes the required position, these operations cannot be followed by an Update.

Procedure

1. Set the operation code to 3.
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 for the key number. To use the system-defined log key (also called system data), specify 125. To use the second system key for system data v2, specify 124.

   When you are performing a non-NCC Update operation immediately following a Get operation, pass the key number exactly as the MicroKernel Engine returned it on the Get operation. Otherwise, the MicroKernel Engine updates the record successfully but returns status code 7 on the first Get operation performed after the update.

Result

If the Update operation succeeds, the MicroKernel Engine 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 Engine releases the lock. However, a multiple-record lock is never released by an Update operation.

If the Update operation fails, the MicroKernel Engine 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 Engine encountered a record-level conflict.

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 the logical current record of an INTEGER 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 logical currency of the application prior to the NCC Update.

Note: The MicroKernel Engine 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 (B_CHUNK_UPDATE) 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

	Op Code	Pos Block	Data Buf	Data Buf Len	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.

Note: Although an extended operation or a Get Key (+50) establishes the required position, you cannot issue an Update Chunk operation immediately after these operations, since they do not return a single record.

Procedure

1. Set the operation code to 53.
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. To use the second system key for system data v2, specify 124.

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 (areas containing [*]): 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.

Element	Length (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.

Element	Length (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.
Chunk Definition (Repeat for each chunk)	12 (for 32-bit applications) 16 (for 64-bit applications)	Each Chunk Definition is a 4-byte Chunk Offset, followed by a 4-byte Chunk Length, followed by a 4-byte User Data for 32-bit applications or an 8-byte User Data for 64-bit applications, 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 65535. However, the chunk definitions must fit in the data buffer. • User Data – (Used only for indirect descriptors.) For 32-bit applications, a 32-bit pointer to the actual chunk data. For 64-bit applications, a 64-bit pointer to the actual chunk data. The format you should use depends on your operating system.[1] The MicroKernel Engine ignores this element for direct chunk descriptor subfunctions.

[1]For DOS 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 Engine 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 for a 32-bit application.

Element	Sample Value	Length (Bytes)
Subfunction	0x8000000	4
NumChunks	3	4
Chunk 0		
Chunk Offset	0x12	4
Chunk Length	0x05	4
User Data	N/A	4
Chunk 1		

Element	Sample Value	Length (Bytes)
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 (areas containing [*]): 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 the following table.

Element	Length (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 65535. However, the chunk definitions must fit in the data buffer.
Distance Between Rows	4	Number of bytes between the beginning of each chunk.
User Data	4 (for 32-bit applications) 8 (for 64-bit applications)	(Used only with indirect descriptors.) For 32-bit applications, a 32-bit pointer to the actual chunk data. For 64-bit applications, a 64-bit pointer to the actual chunk data. The format you should use depends on your operating system.[1] The MicroKernel Engine 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 chunks in the rectangle, as the rectangle is stored in application memory, at the address specified by User Data. The MicroKernel Engine ignores this element for direct rectangle descriptors. However, you must still allocate the element and initialize it to 0.

[1]For DOS 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 application memory with either more or fewer bytes between rows, Application Distance Between Rows allows you to pass that information to the MicroKernel Engine.

When you use an indirect rectangle descriptor, the MicroKernel Engine 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 Engine reads data for the first row from offset 0 of User Data. The MicroKernel Engine reads data in the second row from an address specified by User Data + Application Distance Between Rows. The MicroKernel Engine reads data in the third row from the address specified by User Data + (Application Distance Between Rows * 2), and so on.

The following table shows a sample direct rectangle chunk descriptor structure for a 32-bit application.

Element Name	Sample Value	Length (Bytes)
Subfunction	0x80000002	4
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:

Element	Length (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 Engine calculates the subfunction 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 Engine 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 Engine calculates the offset for all chunks (except the first) by adding the length of the previous chunk to the offset of the previous chunk. 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 subfunction or to the rectangle chunk descriptor subfunction, the MicroKernel Engine calculates the subfunction 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 Engine calculates the offset for all chunks (except the first chunk) based on the record length after the MicroKernel Engine appends the previous chunk.

Result

If the Update Chunk operation succeeds, the MicroKernel Engine 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 Engine 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 Engine releases the lock. However, a multiple-record lock is never released by an Update Chunk operation.

If the Update Chunk operation fails, the MicroKernel Engine 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 Engine encountered a record-level conflict.
97	The data buffer is too small.
103	The chunk offset is too big.
106	The MicroKernel Engine 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 Engine is unpredictable.

Version (26)

For client applications, the Version operation (B_VERSION) returns the local MicroKernel Engine 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 Engine version on that server. For server-based applications, the Version operation returns the server-based MicroKernel Engine version and revision numbers.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	Key Buffer	Key Number
Sent	**X**			**X**		
Returned			**X**	**X**		

Prerequisites

Either the MicroKernel Engine 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 Result.
3. To retrieve the version number of a server-based MicroKernel Engine, you must specify either a valid position block for an opened file on that server or a valid path name in the key buffer.

Result

If you have both a workstation MicroKernel Engine and client Requester configured for access and the Version operation succeeds, the operation returns the version information for the workstation MicroKernel Engine, the client Requester, and the server-based MicroKernel Engine.

Specify a 15-byte data buffer and data buffer length.

If both the client Requester and the workstation MicroKernel Engine are loaded and you specify only a 5-byte data buffer and data buffer length, the operation returns only version information for the client Requester.

If you specify only a 10-byte data buffer and data buffer length, the operation returns the client Requester and the local workstation engine.

If you specify a 15-byte data buffer and data buffer length, the operation returns the client Requester, the local workstation engine, and the server engine (if applicable).

In the data buffer, the Version operation returns a 5-byte Version Block for each MicroKernel Engine or Requester, according to the format shown in the following table. The fifth byte of each block identifies each MicroKernel Engine or Requester.

Element	Length (Bytes)	Description
Version Number	2	Zen version number.
Revision Number	2	Zen revision number.
Requester or Engine Type	1	Type of engine or requester. One of the following: • B (0x42) for the Btrieve engine • C (0x43) for the Client engine • 9 (0x39) for the Workgroup database engine or Linux or macOS database server using Workgroup authentication mode • D (0x44) for DOS workstation • N (0x4E) for client Requester • R (0x52) for Reporting Engine • T (0x54) for Windows server • U (0x55) for Linux, macOS, or Raspbian server using PAM or BTPASSWD authentication

For example, if you are running Zen 14.10 on a Windows server, the Version operation returns the following hexadecimal values in the data buffer:

```
0E 00 0A 00 54
```

After converting these values to decimal, the version number is 14 and the revision number is 10. If the Version operation fails, the MicroKernel Engine returns a nonzero status code.

Positioning

The Version operation has no effect on positioning.

A. Quick Reference of Btrieve Operations

The following table summarizes Btrieve API operations in numerical order by operation code.

Operation	Code	Constant	Description
Open	0	B_OPEN	Makes a file available for access.
Close	1	B_CLOSE	Releases a file from availability.
Insert	2	B_INSERT	Inserts a new record into a file.
Update	3	B_UPDATE	Updates the current record.
Delete	4	B_DELETE	Removes the current record from the file.
Get Equal	5	B_GET_EQUAL	Returns the record whose key value matches the specified key value.
Get Next	6	B_GET_NEXT	Returns the record following the current record in the index path.
Get Previous	7	B_GET_PREVIOUS	Returns the record preceding the current record in the index path.
Get Greater Than	8	B_GET_GT	Returns the record whose key value is greater than the specified key value.
Get Greater Than or Equal	9	B_GET_GE	Returns the record whose key value is equal to or greater than the specified key value.
Get Less Than	10	B_GET_LT	Returns the record whose key value is less than the specified key value.
Get Less Than or Equal	11	B_GET_LE	Returns the record whose key value is equal to or less than the specified key value.
Get First	12	B_GET_FIRST	Returns the first record in the specified index path.
Get Last	13	B_GET_LAST	Returns the last record in the specified index path.
Create	14	B_CREATE	Creates a file with the specified characteristics.

Operation	Code	Constant	Description
Stat	15	B_STAT	Returns file and index characteristics, and number of records.
Extend	16	B_EXTEND	Divides a data file over two logical disk drives. *This operation is not supported in Btrieve 6.0 and later.*
Set Directory	17	B_SET_DIR	Sets the current directory to a specified path name.
Get Directory	18	B_GET_DIR	Returns the current directory for a specified logical disk drive.
Begin Transaction	19 1019	B_BEGIN_TRAN	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	B_END_TRAN	Marks the end of a set of logically related operations.
Abort Transaction	21	B_ABORT_TRAN	Removes operations performed during an incomplete transaction.
Get Position	22	B_GET_POSITION	Returns the position of the current record.
Get Direct/ Chunk	23	B_GET_DIRECT	Returns data from the specified chunks of a record at a specified position.
Get Direct/ Record	23	B_GET_DIRECT	Returns the record at a specified position.
Step Next	24	B_STEP_NEXT	Returns the record from the physical location following the current record.
Stop	25	B_STOP	Terminates the Workgroup MicroKernel Engine. Not available for other instances of MicroKernel Engine.
Version	26	B_VERSION	Returns the version number of the MicroKernel Engine.
Unlock	27	B_UNLOCK	Unlocks a record or records.

Operation	Code	Constant	Description
Reset	28	B_RESET	Releases all resources held by a client.
Set Owner	29	B_SET_OWNER	Assigns an owner name to a file.
Clear Owner	30	B_CLEAR_OWNER	Removes an owner name from a file.
Create Index	31	B_BUILD_INDEX	Creates an index.
Drop Index	32	B_DROP_INDEX	Removes an index.
Step First	33	B_STEP_FIRST	Returns the record in the first physical location in the file.
Step Last	34	B_STEP_LAST	Returns the record in the last physical location in the file.
Step Previous	35	B_STEP_PREVIOUS	Returns the record in the physical location preceding the current record.
Get Next Extended	36	B_GET_NEXT_EXTENDED	Returns one or more records that follow the current record in the index path. Filtering conditions can be applied.
Get Previous Extended	37	B_GET_PREV_EXTENDED	Returns one or more records that precede the current record in the index path. Filtering conditions can be applied.
Step Next Extended	38	B_STEP_NEXT_EXT	Returns one or more successive records from the location physically following the current record. Filtering conditions can be applied.
Step Previous Extended	39	B_STEP_PREVIOUS_EXT	Returns one or more preceding records from the location physically preceding the current record. Filtering conditions can be applied.
Insert Extended	40	B_EXT_INSERT	Inserts one or more records into a file.
Continuous Operation	42	B_CONTINUOUS	Allows system backups without closing active MicroKernel Engine files.

Operation	Code	Constant	Description
Get By Percentage	44	B_SEEK_PERCENT	Returns the record located approximately at a position derived from the specified percentage value.
Find Percentage	45	B_GET_PERCENT	Returns a percentage figure based on the current record's position in the file.
Get Key	+50	KEY_BIAS	Detects the presence of a key value in a file, without returning an actual record.
Update Chunk	53	B_CHUNK_UPDATE	Updates specified portions (chunks) of the current record. This operation can also append data to a record or truncate a record.
Stat Extended	65	B_EXTENDED_STAT	Returns file names and paths of an extended file's components and reports whether a file is using a system-defined log key.
Login/Logout	78	B_LOGIN/B_LOGOUT	Enters user credentials to obtain authentication and authorization tokens from the database engine, or resets the credentials so that they must be entered again.
Get Next Delete Extended	85	B_GET_NEXT_EXT_DELETE	Removes records matching a filter condition, from the logical next position to the end of the file.
Get Previous Delete Extended	86	B_GET_PREV_EXT_DELETE	Removes records matching a filter condition, from the logical previous position to the beginning of the file.
Step Next Delete Extended	87	B_STEP_NEXT_EXT_DELETE	Removes records matching a filter condition, from the physical next position to the end of the file.
Step Previous Delete Extended	88	B_STEP_PREV_EXT_DELETE	Removes records matching a filter condition, Removes records matching a filter condition, from the physical previous position to the beginning of the file.

Operation	Code	Constant	Description
Single-record wait lock	+100	S_WAIT_LOCK	Locks only one record at a time. If the record is already locked, the MicroKernel Engine retries the operation.
Single-record no-wait lock	+200	S_NOWAIT_LOCK	Locks only one record at a time. If the record is already locked, the MicroKernel Engine returns an error status code.
Multiple-record wait lock	+300	M_WAIT_LOCK	Locks several records concurrently in the same file. If the record is already locked, the MicroKernel Engine retries the operation.
Multiple-record no-wait lock	+400	M_NOWAIT_LOCK	Locks several records concurrently in the same file. If the record is already locked, the MicroKernel Engine returns an error status code.
No-wait page lock	+500	NOWRITE_WAIT	In a concurrent transaction, tells the MicroKernel Engine 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).