Pervasive PSQL v11

Btrieve API Guide

Developing Applications Using the Btrieve API

Pervasive Software Inc.
12365 Riata Trace Parkway
Building B
Austin, TX 78727 USA

Telephone: 512 231 6000 or 800 287 4383
Fax: 512 231 6010
Email: database@pervasive.com
Web: http://www.pervasivedb.com

disclaimer

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

trademarks

Btrieve API Guide

January 2013

Contents

Tables

About This Manual

This book is your guide to the Btrieve application programming interface (API).

Who Should Read This Manual

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

Pervasive Software Inc. 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 Pervasive Software Web site.

Manual Organization

This manual is divided into the following sections:

- Chapter 1—Introduction to Btrieve APIs

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

- Chapter 2—Btrieve API Operations

  This chapter documents the Btrieve API operations.

- Appendix A—Quick Reference of Btrieve Operations

  This appendix summarizes the Btrieve API operations in order of operations.

This manual also contains an index.

Typographical Conventions

The documentation uses the following typographical conventions.

Convention	Explanation
bold	Bold typeface usually indicates elements of a graphical user interface, such as menu names, dialog box names, commands, options, buttons, and so forth. Bold typeface is also applied occasionally in a standard typographical use for emphasis.
italics	Italics indicate a variable that must be replaced with an appropriate value. For example, *user_name* would be replaced with an actual user name. Italics is also applied occasionally in a standard typographical use for emphasis, such as for a book title.
cAsE	Uppercase text is used typically to improve readability of code syntax, such as SQL syntax, or examples of code. Case is significant for some operating systems. For such instances, the subject content mentions whether literal text must be uppercase or lowercase.
`monospace`	Monospace text is used typically to improve readability of syntax examples and code examples, to indicate results returned from code execution, or for text displayed on a command line. The text may appear uppercase or lowercase, depending on context.
', ", and “ ”	Straight quotes, both single and double, are used in code and syntax examples to indicate when a single or double quote is required. Curly double quotes are applied in the standard typographical use for quotation marks.
|	The vertical rule indicates an OR separator to delineate items for which you must choose one item or another. See explanation for angle brackets below.
[ ]	Square brackets indicate optional items. Code syntax not enclosed by brackets is required syntax.
< >	Angle brackets indicate that you must select one item within the brackets. For example, <yes | no> means you must specify either “yes” or “no.”
. . .	Ellipsis indicates that the preceding item can be repeated any number of times in succession. For example, [*parameter* . . .] indicates that *parameter* can be repeated. Ellipsis following brackets indicate the entire bracketed content can be repeated.
::=	The symbol ::= means one item is defined in terms of another. For example, a::=b means that item “a” is defined in terms of “b.”
%*string*%	A variable defined by the Windows operating system. *String* represents the variable text. The percent signs are literal text.
$*string*	An environment variable defined by the Linux operating system. *String* represents the variable text. The dollar sign is literal text.

chapter

Introduction to Btrieve APIs 1

The Pervasive PSQL transactional interface is designed for high-performance data handling and improved programming productivity. The transactional interface 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
- ❑ Micro Focus COBOL
- ❑ Microsoft Visual Basic
- ❑ Microsoft Visual C++
- ❑ Watcom C/C++

This chapter discusses the following topics:

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.

Following are the Btrieve API functions.

Table 1 Btrieve API Functions

Function	Operating Systems	Description
BTRV BTRVID	All	Use for complete code portability between operating systems. For most developers, this advantage offsets a very slight performance decrease.

To find the language-specific syntax required when calling a Btrieve API function, refer to the following section in *Pervasive PSQL Programmer's Guide*: Btrieve API Programming.

BTRV Function

BTRV allows an application to make calls to the transactional interface. 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 Function

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

In DOS applications, you must load the DOS Requester with the appropriate /T value. Set /T to equal the number of client IDs you use in the application. For more information about the DOS Requester, refer to Pervasive's *Getting Started With Pervasive PSQL* manual.

BTRCALL Function

For Windows and Linux, BTRCALL and BTRCALL32 are 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 Function

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

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

BTRCALLID32 Function

The BTRCALLID32 function is the same as the BTRCALLID function.

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 MicroKernels.

Btrieve API Function Parameters

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

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

The parameters to the Btrieve API functions are as follows:

- Operation Code
- Status Code (BASIC and COBOL only)
- Position Block
- Data Buffer
- Data Buffer Length
- Key Buffer
- Key Number
- Client ID (BTRVID and BTRCALLID functions only)
- Key Length (BTRCALL, BTRCALLID, BTRCALL32, and BTRCALLID32 functions only)

Operation Code

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

Note *C developers:* The variable you specify must be BTI_WORD (an unsigned short integer), and is passed by value.

Status Code

The transactional interface returns status codes as 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. This parameter is a 2-byte integer containing a coded value that indicates whether any errors occurred during the operation. After a Btrieve API call, the application must always check the value of the status variable to determine if the call was successful.

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

- API Success
- Anticipated API failure
- Unanticipated API failure

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

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

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

Position Block

The Position Block parameter is the address of a 128-byte array that the transactional interface 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 transactional interface 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 transactional interface uses the Position Block to determine which file a particular call is for. Similarly, when you open the same file more than once, the transactional interface uses a different Position Block for each separate Open operation. Likewise, the transactional interface uses a different Position Block for each separate client that opens the same file. Multiple clients cannot share position blocks.

Data Buffer

Your application transfers data to and from a file using the Data Buffer. The information passed to or from the transactional interface 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, transactional interface 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, transactional interface modification operations may destroy data following the Data Buffer.

Data Buffer Length

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

Note *BASIC developers:* Your application must pass the Data Buffer Length parameter ByRef as a Long integer.

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

When you are inserting records into or updating a file with variable-length records, the Data Buffer Length should equal the record length specified when you first created the file, plus the number of characters included beyond the fixed-length portion. When you are retrieving variable-length records, the Data Buffer Length should be large enough to accommodate the longest record in the file. If a record is longer than 64 KB, you must use a chunk operation to operate on a portion of the record.

The transactional interface 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 transactional interface to overwrite memory. The Data Buffer Length should always represent the size of the allocated Data Buffer.

Key Buffer

Your application must pass the Key Buffer parameter on every Btrieve 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 transactional interface returns an error if the string variable passed as the Key Buffer is shorter than the key's defined length. If your application's first call does not require initialization of the Key Buffer, assign the string variable the value SPACE$(*x*), where *x* represents the key's defined length. Until your application assigns some value in BASIC to the string variable, it has a length of 0.

C developers: Your application must pass the Key Buffer as the address of a variable containing the key value. The file BTITYPES.H defines the Key Buffer as a VOID pointer (BTI_VOID_PTR). Your application can then define the Key Buffer type as needed.

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

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

In most environments, the transactional interface cannot determine the Key Buffer length when an application makes a Btrieve API call. Therefore, you must ensure that the buffer is at least as long as the key length you specified when you first created the key. Otherwise, Btrieve API operations may destroy data stored in memory following the Key Buffer. It is best to always have a 255-byte Key Buffer, because 255 is the maximum length for a key.

Key Number

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

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

Client ID

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

Table 2 Client ID Structure

Element	Length (bytes)	Description	
Filler	12	Initialize to 0.	
Service Agent ID	2	Identifies each instance of your application to the transactional interface. 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 transactional interface assumes special meaning for the following values:	
		0x4140 (@A)	Used internally.
		0xFFFF	Used internally.
		0x4952 (RI)	Used internally.
		0x5244 (DR)	Used internally.
		0x4553 (SE) 0x4353 (SC) 0x4344 (DC) 0x4544 (DE) 0x5544 (DU)	Used to identify clients originated by Scalable SQL.

Table 2 Client ID Structure continued

Element	Length (bytes)	Description	
		0x5257 (WR)	Used by Btrieve Requesters.
Client Identifier	2	Establishes a client's identity within the current instance of your application. The transactional interface uses this unique identifier for concurrency and transaction-processing purposes.	

Key Length

The Key Length parameter is used only in the BTRCALL, BTRCALLID, BTRCALL32, and BTRCALLID32 functions.

When using these functions, you must pass the Key Length parameter as an unsigned char (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).

Summary of Btrieve API Operations

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

Session-Specific Operations

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

Table 3 Session-Specific Operations

Operation	Code	Description
Stop	25	Terminates the workstation transactional interface (not available for server-based transactional interface).
Version	26	Returns the version number of the transactional interface.
Reset	28	Releases all resources held by a client.
Begin Transaction	19 1019	Marks the beginning of a set of logically related operations. Operation 19 begins an exclusive transaction. Operation 1019 begins a concurrent transaction.
End Transaction	20	Marks the end of a set of logically related operations.
Abort Transaction	21	Removes operations performed during an incomplete transaction.

File-Specific Operations

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

- File access and information. These operations allow you to create a file, open and close a file, retrieve file statistics, set and clear the file's owner name, start or stop continuous operation mode on a file, unlock a file, and create and drop indexes on a file.
- **Data retrieval.** These operations allow you to retrieve a single record or a set of records given specified criteria. The Btrieve API supports data retrieval either by logical location in an index path or by physical location. For more information, refer to "Accessing Records" in the *Pervasive PSQL Programmer's Guide*.

  In addition, you can apply biases to the operation codes to control file and record locking in multi-client situations. For more information, refer to "Supporting Multiple Clients" in the *Pervasive PSQL Programmer's Guide*.
- Data manipulation. These operations allow you to insert, update, or delete data.

Table 4 File Access and Information Operations

Operation	Code	Description
Open	0	Makes a file available for access.
Close	1	Releases a file from availability.
Create	14	Creates a file with the specified characteristics.
Stat	15	Returns file and index characteristics, and number of records.
Stat Extended	65	Returns file names and paths of an extended file's components and reports whether a file is using a system-defined log key.
Set Owner	29	Assigns an owner name to a file.
Clear Owner	30	Removes an owner name from a file.
Unlock	27	Unlocks a record or records.

Table 4 File Access and Information Operations continued

Operation	Code	Description
Create Index	31	Creates an index.
Drop Index	32	Removes an index.

Table 5 Data Retrieval Operations

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

Table 5 Data Retrieval Operations continued

Operation	Code	Description
Get Key	+50	Detects the presence of a key value in a file, without returning an actual record.
Get By Percentage	44	Returns the record located approximately at a position derived from the specified percentage value.
Find Percentage	45	Returns a percentage figure based on the current record's position in the file.
Non-Index-Based (Physical) Retrieval		
Get Position	22	Returns the position of the current record.
Get Direct/Chunk	23	Returns data from the specified portions (chunks) of a record at a specified position.
Get Direct/ Record	23	Returns the record at a specified position.
Step Next	24	Returns the record from the physical location following the current record.
Step First	33	Returns the record in the first physical location in the file.
Step Last	34	Returns the record in the last physical location in the file.
Step Previous	35	Returns the record in the physical location preceding the current record.
Step Next Extended	38	Returns one or more successive records from the location physically following the current record. Filtering conditions can be applied.
Step Previous Extended	39	Returns one or more preceding records from the location physically preceding the current record. Filtering conditions can be applied.
Get By Percentage	44	Returns the record located approximately at a position derived from the specified percentage value.
Find Percentage	45	Returns a percentage figure based on the current record's position in the file.

Table 5 Data Retrieval Operations continued

Operation	Code	Description
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 transactional interface 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 transactional interface returns an error status code.
No-wait page write lock	+500	In a concurrent transaction, tells the transactional interface 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).

Table 6 Data Manipulation Operations

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

Unsupported Operations

When looking at transactional interface 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 Pervasive PSQL and you should not use them in your applications. The following operations are not supported.

Table 7 Unsupported Operations

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

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 operation (0) 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 transactional interface 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.

chapter

2

Btrieve API Operations

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

- Name, code, and description of the operation.
- Parameters—a table indicating which of the six parameter values the operation expects from and returns to your application. A "Sent" parameter is sent from the application to the operation; a "Returned" parameter is returned from the operation to the application when the operation is complete.
- Prerequisites—the conditions your application must satisfy for the operation to be successful.
- Procedure—the steps for initializing the parameters that the operation requires.
- Details—additional information about the operation.
- Result—the results of both a successful and an unsuccessful operation. Each operation returns a status code, informing your application of the outcome of the operation. Status Code 0 indicates the operation was successful. A nonzero status code usually indicates a failure. However, some nonzero status codes are informative and appear even when the associated operation is successful—for example, Status Code 60 means the specified reject count has been reached.
- Positioning—the effect the operation has on the logical and/or physical currency of the records in a file.

This chapter includes the following Btrieve API operations, organized alphabetically:

- Create (14)
- Create Index (31)
- Delete (4)
- Drop Index (32)
- End Transaction (20)
- Find Percentage (45)
- Get By Percentage (44)
- Get Direct/Chunk (23)
- Get Direct/Record (23)
- Get Directory (18)
- Get Equal (5)
- Get First (12)
- Get Greater (8)
- Get Key (+50)
- Get Last (13)
- Get Less Than or Equal (11)
- Get Next (6)
- Get Next Extended (36)
- Get Position (22)
- Get Previous (7)
- 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 Extended (38)

- Step Previous (35)
- Step Previous Extended (39)
- Stop (25)
- Unlock (27)
- Update (3)
- Update Chunk (53)
- Version (26)

Abort Transaction (21)

The Abort Transaction operation (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	✓					
Returned						

Prerequisites

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

Procedure

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

Result

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

If the Abort Transaction operation is unsuccessful, the transactional interface 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 transactional interface does not permanently complete any operations in the set unless you request the completion with an explicit End Transaction operation (20). Changes made within a transaction are not visible to other users until the End Transaction operation is successfully performed.

The transactional interface prohibits certain operations during transactions because they have too great an effect on the file or on performance. These operations include Set Owner, Clear Owner, Create Index, and Drop Index.

Parameters

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

Prerequisites

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

Procedure

Set the Operation Code to 19 to begin an exclusive transaction, or 1019 to begin a concurrent transaction. The transactional interface 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 transactional interface 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.

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

Result

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

If the Begin Transaction operation is unsuccessful, the transactional interface 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 transactional interface decrypts the file during a Clear Owner operation.

Parameters

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

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 transactional interface 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 transactional interface decrypts the data during a Clear Owner operation. The more data that was encrypted, the longer the Clear Owner operation takes.

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

3	The file is not open.
41	The transactional interface 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 operation (0) for that file.

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

Note When you close a file inside a transaction, the transactional interface 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	✓	✓				
Returned						

Prerequisites

- The file must be open.

Procedure

1. Set the Operation Code to 1.
2. Pass a valid Position Block for the file to close.

Result

If the Close operation is successful, the Position Block for the closed file is no longer valid.

If the Close operation fails, the file remains open and the transactional interface 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 transactional interface 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 transactional interface 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	✓		✓	✓		✓
Returned			✓	✓		

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

Procedure

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

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

a. Set the Operation Code to 42.

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

The following example is for Windows 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 filenames in the Data Buffer parameter as described in step 1b. Also, place the length of the name (or names) in the Data Buffer Length parameter as described in step 1c.

Details

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

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

- The presence of duplicate filenames in the Data Buffer does not affect how the Continuous Operation operation works. The MKDE places the specified file in continuous operation mode only once.
- 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 corresponding file's name with ".^^^" for the file name extension. The transactional interface 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 SRDE. Files related by referential integrity constraints should be passed in on a single continuous operations call.

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

Result

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

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

11	The specified filename is invalid.
12	The transactional interface cannot find the specified file.
41	The transactional interface 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 transactional interface returns a nonzero status code, the Continuous Operation operation returns in the Data Buffer the portion of the input string that generated the error. If no input string was used, the Data Buffer contains the file names that caused the

error. The Data Buffer Length reflects the length of the output string in the Data Buffer. At this point, the Data Buffer Length contains the length of that filename.

Positioning

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

Create (14)

The Create operation (B_CREATE) lets you generate 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 the database engine uses the file name for various areas of functionality while ignoring the file name extension. Since only the file name is used to differentiate files, files that differ only in their file name extension look identical to the database engine.

Parameters

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

Prerequisites

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

Procedure

1. Set the Operation Code to 14.
2. Specify the file specifications, key specifications, and any alternate collating sequences in the Data Buffer as described in Details. (All the values for the file specifications and key specifications you store in the Data Buffer must be in binary format.)
3. Specify the Data Buffer Length. This is the length of the buffer that contains the Create specifications, not the length of the records in the file.

4 Specify the pathname for the file in the Key Buffer. Be sure to terminate the pathname with a blank or binary zero. The pathname can be up to 255 characters long, including the volume name and the terminator.

   For details about path names supported by Pervasive PSQL clients, see Network Path Formats Supported by Pervasive Requesters in *Getting Started With Pervasive PSQL*. See also Database URIs in *Pervasive PSQL Programmer's Guide*.

5 Specify a value for the Key Number parameter, using one of the values in Table 14.

Details

Table 8 illustrates the order in which the file and key specifications must be stored.

Table 8 Data Buffer Structure for Create Operation

Description		Data Type[1]	Byte #	Example Value[2]
File Specification				
Logical Fixed Record Length[3]		Short Int[4]	0, 1	Size of all fields combined.
Page Size.	Pervasive PSQL 6.x through 9.4	Short Int	2, 3	512
	Pervasive PSQL 6.x and later			1,024
	Pervasive PSQL 6.x through 9.4			1,536
	Pervasive PSQL 6.x and later			2,048
	Pervasive PSQL 6.x through 9.4			3,072
				3,584
	Pervasive PSQL 6.x and later			4,096
	Pervasive PSQL 9.0 through 9.4x			8,192
	Pervasive PSQL 9.5 and later			16,384
	A minimum size of 4096 bytes works best for most files. If you want to fine-tune this, refer to Creating a File with Page Level Compression for more information. When creating 9.5 file format files, if the logical page size specified is 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 the older file formats.			

Table 8 Data Buffer Structure for Create Operation continued

Description	Data Type[1]	Byte #	Example Value[2]	
Number of Keys (Indexes)	Byte	4		
File Version	Byte	5	0x60	Version 6.0
			0x70	Version 7.0
			0x80	Version 8.0
			0x90	Version 9.0
			0x95	Version 9.5
			0x00	Use database engine default
Reserved. (Not used during a Create operation.)	Reserved	6- 9	0	
File Flags. Specifies the file attributes. The example file does not use any.	Short Int	10, 11	0	
Number of Extra Pointers. Sets the number of duplicate pointers to reserve for future key additions. Used if the file attributes specify Reserve Duplicate Pointers.	Byte	12	0	
Reserved. (Not used during a Create operation.)	Reserved	13	0	
Preallocated Pages. Sets the number of pages to preallocate. Used if the file attributes specify Page Preallocation.	Short Int	14, 15	0	
Key Specification for Key 0 (Last Name)				
Key Position. Provides the position of the first byte of the key within the record. The first byte in the record is 1.	Short Int	16, 17	1	
Key Length. Specifies the length of the key, in bytes.	Short Int	18, 19	25	
Key Flags. Specifies the key attributes.	Short Int	20, 21	EXTTYPE_KEY + NOCASE_KEY + DUP + MOD	
Not Used for a Create.	Byte	22-25	0	
Extended Key Type. Used if the key flags specify Use Extended Key Type. Specifies one of the extended data types.	Byte	26	ZSTRING	

Table 8 Data Buffer Structure for Create Operation continued

Description	Data Type[1]	Byte #	Example Value[2]
Null Value (legacy nulls only). Used if the key flags specify Null Key (All Segments) or Null Key (Any Segment). Specifies an exclusion value for the key. See Null Value for more conceptual information on legacy nulls and true nulls.	Byte	27	0
Not Used for a Create.	Byte	28, 29	0
Manually Assigned Key Number. Used if the file attributes specify Key Number. Assigns a key number.	Byte	30	0
ACS Number. Used if the key flags specify Use Default ACS, Use Numbered ACS in File, or Use Named ACS. Specifies the ACS number to use.	Byte	31	0
Key Specification for Key 1 (Employee ID)			
Key Position. (Employee ID starts at first byte after Middle Initial.)	Short Int	32, 33	52
Key Length.	Short Int	34, 35	4
Key Flags.	Short Int	36, 37	EXTTYPE_KEY + MOD
Not Used for a Create.	Byte	38-41	0
Extended Key Type.	Byte	42	INTEGER
Null Value.	Byte	43	0
Not Used for a Create.	Byte	44, 45	0
Manually Assigned Key Number.	Byte	46	0
ACS Number.	Byte	47	0
Key Specification for Page Compression			

Table 8 Data Buffer Structure for Create Operation continued

Description	Data Type[1]	Byte #	Example Value[2]
Physical Page Size[5]	Char	A	512 (default value)

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

[2]For simplification, the non-numeric example values are for C applications.

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

[4]Short Integers (Short Int) must be stored in the "Little Endian" byte order, which is the Low To High ordering of Intel-class computers.

[5]Only used with page-level compression. Must be used in conjunction with the Page Compression file flag (see Table 6). See Creating a File with Page Level Compression for more information.

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

File Specification

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

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

Page Size. (Offset 0x02) Page size is determined by your file format version. See Choosing a Page Size in *Pervasive PSQL Programmer's Guide*.

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

File Version. (Offset 0x05) The transactional interface file version to be created. In prior releases, the transactional interface used a two byte integer to receive the number of indexes on a create operation. The high order byte of this integer was always zero because the

maximum number of indexes is 119. This high-order byte has always been used in the Stat operation to return the file version and it can now be used to specify the file version in the Create operation without breaking any previous applications. The supported file versions for Create are 6.x, 7.x, 8.x, 9.x, 9.5 which are represented in the specified byte with hex values 0x50, 0x60, 0x70, 0x80, 0x90, 0x95, respectively.

Physical Page Size. (Offset 0xA). This field was previously marked "Not Used." This field is used in conjunction with the "Page Compression" file flag. If the "Page Compression" flag is not specified then this field is ignored.

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 zero is specified the engine will use a default value for the physical page size. The default value is 512 bytes.

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 *Pervasive PSQL Programmer's Guide*

File Flags. (Offset 0x0A) The bit settings in the File Flags word specify file attributes. Table 9 shows the binary, hexadecimal, and decimal representations of the file flag values.

Table 9 File Flag Values

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

Table 9 File Flag Values continued

Attribute	Constant	Binary	Hex	Decimal
Use Page Compression[2]	PAGE_COMPRESSED	0010 0000 0000 0000	2000	8,192

[1]If you do not explicitly specify whether to include system data in the file, the Btrieve API uses the current setting of the transactional interface configuration option “Include System Data.”

[2]Only used with page level compression. Used in conjunction with the Physical Page Size key specification. See *Pervasive PSQL Programmer's Guide* Creating a File with Page Level Compression for more information.

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 transactional interface 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 transactional interface ignores the Variable Length Records bit. If you set the Key-only File bit, the transactional interface ignores the System Data bits.

The database engine automatically uses data compression on files that use system data and have a record length that exceeds the maximum length allowed. See Table 12 in *Pervasive PSQL Programmer's Guide*.

Set the Duplicate Pointers bit if you anticipate adding an index sometime after creating a file, and if that index has many duplicate values. Setting this bit causes the transactional interface 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 transactional interface 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 transactional interface does not require consecutive key numbers; files can have gaps between key numbers. When a key is created, by default the transactional interface assigns the lowest available key number to that index (beginning with 0). However, some applications may require a key number different from the default assignment.

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

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

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

Key Specification Blocks

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

The maximum number of key segments allowed depends on the file's page size.

Page Size (bytes)	Maximum Key Segments byFile Version		
	8.x and prior	9.0	9.5
512	8	8	rounded up[2]
1,024	23	23	97
1,536	24	24	rounded up[2]
2,048	54	54	97
2,560	54	54	rounded up[2]
3,072	54	54	rounded up[2]
3,584	54	54	rounded up[2]
4,096	119	119	204 (or 119)[3]
8,192	n/a[1]	119	420 (or 119)[3]
16,384	n/a[1]	n/a[1]	420 (or 119)[3]

[1]"n/a" stands for "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 1,024, 2,560 is rounded up to 4,096, and so forth.

[3]The maximum number of index segments that can be used with the relational interface is 119. For the transactional interface, the maximum number is 204 for a page size of 4,096, and 420 for page sizes 8,192 and 16,384.

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*.

Key Position. (Offset 0x00) 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. (Offset 0x02) The length of the key or key segment. The maximum length of a key, including all key segments, is 255 bytes.

Key Flags. (Offset 0x04) The bit settings in the Key Flags word specify key attributes. Table 10 shows the binary, hexadecimal, and decimal representations of the Key Flags values.

Table 10 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	1	1
Modifiable Key Values	MOD	0000 0000 0000 0010	2	2
Use Old Style BINARY Data Type	BIN	0000 0000 0000 0100	4	4
Use Old Style STRING Data Type (bits 2 and 8 must be 0)		0000 0000 0000 0000	0	0
Null Key (All Segments)	NUL	0000 0000 0000 1000	8	8
Segmented Key	SEG	0000 0000 0001 0000	10	16
Use Default ACS	ALT	0000 0000 0010 0000	20	32
Use Numbered ACS in File	NUMBERED_ACS	0000 0100 0010 0000	420	1,056
Use Named ACS	NAMED_ACS	0000 1100 0010 0000	C20	3,104
Descending Sort Order	DESC_KEY	0000 0000 0100 0000	40	64
Repeating Duplicates (combine with attribute DUP)	REPEAT_DUPS_ KEY	0000 0000 1000 0000	80	128
Use Extended Data Type	EXTTYPE_KEY	0000 0001 0000 0000	100	256
Null Key (Any Segment)	MANUAL_KEY	0000 0010 0000 0000	200	512
Case Insensitive Key	NOCASE_KEY	0000 0100 0000 0000	400	1,024

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

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

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

- All segments of the same key must have the same Duplicate Key Values, Repeating Duplicates, Modifiable Key Values, and Null Key values. (If you specify the legacy Null Key attribute, either All Segments or Any Segment, you can assign different null values for individual segments.)
- All segments of the same key must use the same ACS.
- Individual segments of the same key can have different Descending Sort Order and Extended Data Type values.

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

Extended Data Type. (Offset 0x0A) You specify the Extended Data Type in byte 10 of the Key Specification block as a binary value. Table 11 shows the codes for the extended data types.

Table 11 Extended Data Types

Type	Code	Type	Code
CHAR	0	ZSTRING	11
INTEGER	1	UNSIGNED BINARY	14
FLOAT	2	AUTOINCREMENT	15
DATE	3	NUMERICSTS	17
TIME	4	NUMERICSA	18
DECIMAL	5	CURRENCY	19
MONEY	6	TIMESTAMP	20
LOGICAL	7	WSTRING	25
NUMERIC	8	WZSTRING	26
BFLOAT	9	GUID	27
LSTRING	10	NULL INDICATOR SEGMENT	255

Extended data type codes 12, 13, 16, and 21 through 24 are reserved for future use.

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

Regarding the data type you assign to a key, transactional interface does not ensure that the records you input adhere to the data types defined for the keys. For example, you could define a TIMESTAMP key in a file, but store a character string there. Your Btrieve API application would work fine, but an ODBC application that tries to access the same data using the ODBC TIMESTAMP format might fail, because the byte format could be different and the algorithms used to generate the timestamp value could be different. For complete descriptions of the data types, refer to *SQL Engine Reference*.

Null Value. (Offset 0x0B) of the Key Specification block represents an exclusion value for the key. If you have defined a key as a null key, you must supply a value for the transactional interface 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 the following topic in *Pervasive PSQL Programmer's Guide*: Null Value.

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

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

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

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

Alternate Collating Sequence

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

User-Defined ACSs. To create an ACS that sorts string values differently from the ASCII standard, your application must place 265

bytes directly into the Data Buffer, using the format displayed in Table 12.

Table 12 Data Buffer for Creating a User-Defined ACS

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

For examples of user-defined ACSs, refer to the following topic in *Pervasive PSQL Programmer's Guide*: Alternate Collating Sequences.

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

Table 13 Data Buffer for Specifying an ISR ACS

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

Data Buffer Length

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

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

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

Key Number

The Create operation's Key Number parameter is used to determine if the transactional interface warns you when a file of the same name already exists, and also to determine whether the transactional interface should use a local or remote engine when creating the file.

Use Table 14 to determine which value you should use for the Key Number parameter.

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

Table 14 Key Number Parameter for Create Operation

CREATE Operation	No preference	Force local engine to create file	Force remote engine to create file
Normal create (overwrite file if it 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.

In releases prior to Pervasive.SQL v8.5, it was always possible to manipulate transactional interface files through the operating system, because the transactional interface depended on the rights and privileges given by the operating system to the Pervasive PSQL user.

Now, if you have a secure Pervasive PSQL database, these operating system access rights may have been removed in the process of securing the database from unauthorized access. This makes programmatically deleting or moving the files difficult since the rights required 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 you set up the Create operation to use the rename or delete subfunctions.

Table 15 Create Operation 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 transactional interface 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 *Pervasive PSQL Programmer's Guide*.

The security information is processed just 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.

Table 16 Create Subfunctions - Use of URI Parameters in Key Buffer

Function	URI parameter file=	URI parameter dbfile=	URI parameter table=
Rename	✓	✓	✗
Delete	✓	✓	✗

Table 17 Create Subfunctions - Use of URI Parameters in Data Buffer

Function	file=	dbfile=	table=
Rename	✓	✓	✗
Delete	Not applicable	Not applicable	Not applicable

Notes on Rename and Delete Subfunctions

- The previous functionality of the Create operation is intact. Follow the existing documentation on the Create operation if you want to create a new transactional interface data file.
- 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 is successful, the transactional interface either warns you of the existence of a file with the same name or creates the new file according to your specifications. The new file does not contain any records. The Create operation does not open the file. Your application must perform an Open operation before it can access the file.

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

2	The application encountered an I/O error.
11	The specified file name is invalid.
18	The disk is full.
22	The data buffer length is too short.
24	The page size or data buffer size is invalid.
25	The application cannot create the specified file.
26	The number of keys specified is invalid.
27	The key position is invalid.

28	The record length is invalid.
29	The key length is invalid.
48	The alternate collating sequence definition is invalid.
49	The extended data type is invalid.
59	The specified file already exists.
104	The transactional interface does not recognize the locale.
105	The file cannot be created with Variable-tail Allocation Tables (VATs).
134	The transactional interface 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	✓	✓	✓	✓		✓
Returned						

Prerequisites

- The file must be open.
- The number of existing key segments in the file must be less than or equal to maximum number of key segments allowed minus the number of key segments to be added.
- The maximum number of key segments allowed depends on the file's page size. The following table shows these values:

Page Size (bytes)	Maximum Key Segments byFile Version		
	8.x and prior	9.0	9.5
512	8	8	rounded up[2]
1,024	23	23	97
1,536	24	24	rounded up[2]
2,048	54	54	97
2,560	54	54	rounded up[2]
3,072	54	54	rounded up[2]
3,584	54	54	rounded up[2]
4,096	119	119	204 (or 119)[3]
8,192	n/a[1]	119	420 (or 119)[3]

Page Size (bytes)	Maximum Key Segments byFile Version		
	8.x and prior	**9.0**	**9.5**
16,384	n/a[1]	n/a[1]	420 (or 119)[3]

[1]"n/a" stands for "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 1,024, 2,560 is rounded up to 4,096, and so forth.

[3]The maximum number of index segments that can be used with the relational interface is 119. For the transactional interface, the maximum number is 204 for a page size of 4,096, and 420 for page sizes 8,192 and 16,384.

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 16-byte key specification block in the Data Buffer. Use the same structure as defined in Table 8. Store the information for the key position and the key length as integers. If you are rebuilding the system-defined log key (also called system data), the Data Buffer must be at least 16 bytes long and initialized to zeroes.
4. To define an ACS for the new key, perform one of the following steps:
   - To use the default ACS, which is the first ACS already defined in the file, specify the Use Default ACS attribute in the Key Flags word.
   - To define a new ACS, specify the Use Numbered ACS attribute in the Key Flags word and set the ACS Number field to zero (0). In addition, store the 265-byte ACS after the last key specification block in the Data Buffer.

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

ACS Type	Length (in 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 * (# of segments)

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

16 * (# of segments) + 265

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

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 file's page size.)

Details

The transactional interface allows you to assign specific key numbers when creating a key. This capability complements the ability to delete a key and not have the transactional interface renumber all keys that have a key number greater than that of the deleted key. If an application drops an index and instructs the transactional interface 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 transactional interface first checks for an existing ACS (using the name you specified) before adding it to the file. If the transactional interface finds an existing ACS with the name you specified, the transactional interface 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 transactional interface 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 transactional interface and a client initiates a Create Index process, remote clients can perform Get and Step operations on the same file while the transactional interface 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 transactional interface does not have to change the images of the records in order to create the key.

However, if the key being created is an AUTOINCREMENT key, the transactional interface 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 transactional interface returns Status Code 84 if a client attempts to create an AUTOINCREMENT key while another client has locked a record. Similarly, the transactional interface 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 transactional interface immediately adds the new key to the file. The time required for this operation depends on the total number of records to be indexed, the size of the file, and the length of the new index.

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

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

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

If the Create Index operation is unsuccessful, the transactional interface 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 transactional interface can return the following status codes:

22	The data buffer length is too short.
27	The key position is invalid.
41	The transactional interface 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 transactional interface does not recognize the locale.
134	The transactional interface cannot read the International Sorting Rule.

135	The specified International Sort Rule table is corrupt or otherwise invalid.
136	The transactional interface cannot find the specified Alternate Collating Sequence in the file.

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

Positioning

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

Delete (4)

The Delete operation (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	✓	✓				
Returned		✓				

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

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

The transactional interface does not allow Delete operations after a Get Key operation (+50). Before the transactional interface 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 transactional interface 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 transactional interface cannot perform its passive concurrency conflict checking without the compare. When the Delete fails, the transactional interface returns Status Code 8.

Result

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

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

8	The current positioning is invalid.
80	The transactional interface 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 re-used when records are added in the future. Recovering disk space can only be done by re-creating the file and inserting all the records into it.

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	✓	✓				✓
Returned						

Prerequisites

- The file must be open.
- The key must exist in the file.
- No transactions can be active.

Procedure

1. Set the Operation Code to 32.
2. Pass the Position Block of the file that contains the key to drop.
3. Store the number of the key to drop in the Key Number parameter. To drop the system-defined log key (also called system data), specify 125.

Details

If you drop the system-defined log key, you can rebuild it using the Create Index (31) operation.

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

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

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

Result

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

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

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

If processing is interrupted while the transactional interface is dropping an index, you can access the data in the file by the file's other keys. The transactional interface 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	✓					
Returned						

Prerequisites

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

Procedure

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

Result

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

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

38	The transactional interface 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 the Get By Percentage operation (44). Find Percentage finds the approximate position of a record either relative to a key path or as the record's physical location within the file. The position is expressed as a percentage value. See the "Result" section for a definition of the range of percentage values.

Parameters

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

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

Prerequisites

- The file must be open.
- If you are seeking the percentage relative to a key path, the file cannot be a data-only file.
- When you are seeking the percentage by a record's physical location in the file, you must provide the 4-byte physical location of the record in the Data Buffer parameter. You can retrieve this location with a Get Position operation (22).

Procedure

1. Set the Operation Code to 45.
2. Pass the Position Block for the file.

3 If you are seeking the percentage relative to the record's physical location within the file, store the record's physical address in the Data Buffer parameter. 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 bytes. (This 4-byte minimum is a requirement of the transactional interface's internal implementation). If specifying a granularity for the search, set the data buffer length to a minimum of 12 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:

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

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

Details

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

To optimize the Find Percentage operation, the transactional interface 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 prior to Pervasive PSQL 9, this value was always 10,000.

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

➤ To specify a granularity in the Find Percentage operation

1. Place ExPc in the second four bytes of the data buffer.
2. Place the desired granularity in the third four bytes 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 FindPercentage with 100 as the percentage, and 365 as the granularity.

Result

If the Find Percentage operation is successful, the transactional interface 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 10,000 (100.00 percent). Note this is not the physical or logical position.

The percentage value is returned as a 2-byte integer in low-byte, high-byte order. For example:

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

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

If the Find Percentage operation is unsuccessful, the transactional interface 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 transactional interface does not allow the attempted operation.
43	The specified record address is invalid.
82	The transactional interface 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 that record's relative position in the file, where the position is based on a percentage value you supply when you call the operation. You must also specify whether the position is relative to a specified key path or represents the record's actual physical location in the file.

Parameters

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

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

Prerequisites

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

Procedure

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

- +100—Single wait record lock.
- +200—Single no-wait record lock.
- +300—Multiple wait record lock.
- +400—Multiple no-wait record lock.

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

2. Pass the Position Block for the file.
3. Store the percentage value as a 2-byte integer in the Data Buffer. See the "Details" section for the acceptable range of percentage values and related information.
4. Set the Data Buffer Length to a value greater than or equal to the length of the largest possible record that could be returned. (The transactional interface's 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.
    a. If you are seeking the record by a key path, set the Key Number parameter to the actual key number. To use the system-defined log key (also called system data), specify 125.
    b. If you are seeking the record by the record's physical position in the file, set the Key Number parameter to -1 (0xFF).

Details

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 10,000 (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 5,000 (0x1388). After byte-swapping 0x1388, store 0x88 and 0x13 in the first two bytes of the Data Buffer parameter.

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.

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 transactional interface 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 prior to Pervasive PSQL 9, this value was always 10,000.

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

➤ To specify a granularity in the Get By Percentage operation

1. Place ExPc in the second four bytes of the data buffer.
2. Place the desired granularity in the third four bytes 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 GetByPercentage with 100 as the percentage, and 365 as the granularity.

Result

If the Get By Percentage operation is successful, the transactional interface 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 transactional interface 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 transactional interface 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 transactional interface 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 transactional interface 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 is unsuccessful, the transactional interface 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 transactional interface does not allow the attempted operation.
43	The specified record address is invalid.
82	The transactional interface lost positioning.

Positioning

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

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

If the Get By Percentage operation is unsuccessful, the transactional interface 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 65,535 bytes. Such records are too long to be retrieved by the other Get and Step operations, due to restrictions on the length of the Data Buffer parameter. Your application specifies the record from which chunks are to be retrieved by supplying its physical address. The location of a chunk in a record is generally specified by its offset and length.

Parameters

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

Prerequisites

- The file must be open.
- You must provide the 4-byte physical location of the record. You can retrieve this location with a Get Position operation (22).
- You must provide a large enough Data Buffer to contain all values that a Get Direct/Chunk operation returns. The Data Buffer must also be able to contain the entire chunk descriptor (all the chunk definitions) when the Get Direct/Chunk operation is performing an indirect chunk operation. The maximum size of the Data Buffer is limited as shown in Table 18.

Table 18 Data Buffer Size Limitations by Environment

Environment	Maximum Data Buffer Size
Local calls to server or workstation engine	64,512 bytes
Remote calls via DOS Requester (BREQNT)	55,512 bytes
Remote calls via Windows or Linux Requester	65,153 bytes

Procedure

1. Set the Operation Code to 23. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.
3. Specify a Data Buffer, as described in Details.
4. Specify the Data Buffer Length as either the length of the input structure (Table 19 or Table 20) or the number of bytes you requested for the transactional interface to retrieve, whichever is larger.

   Some options for the Get Direct/Chunk operation retrieve chunks to locations other than the Data Buffer. See the Details section for more information about calculating the Data Buffer Length.
5. Set the Key Number parameter to -2 (0xFE).

Details

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

- Random Chunk Descriptor—To retrieve a single chunk per operation, or to retrieve multiple chunks in a single operation when the chunks are spaced randomly throughout the record.
- Rectangle Chunk Descriptor—To retrieve multiple chunks in an operation, when each chunk is the same length and chunks are spaced equidistantly in the record.

Random Chunks

The following example shows a record with three randomly spaced chunks (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	[*]	[*]	[*]	[*]	[*]	[*]	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	[*]	[*]	[*]	[*]	[*]	[*]
[*]	[*]	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	4F

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

Table 19 Data Buffer for Random Chunk Operations

Element	Length (Bytes)	Description
Record Address	4	The 4-byte physical location of the record. You can retrieve this location with a Get Position operation (22).
Random Chunk Descriptor		
Subfunction	4	Type of chunk descriptor; one of the following: • 0x80000000 (Direct random chunk descriptor)—Retrieves chunks directly into the Data Buffer. The first chunk is retrieved and stored at offset 0 in the Data Buffer, the second chunk immediately follows the first, and so on. • 0x80000001 (Indirect random chunk descriptor)—Retrieves chunks into addresses specified by the Chunk Definitions.
NumChunks	4	Number of chunks to retrieve. The value must be at least 1. Although no explicit maximum value exists, the chunk descriptor must fit in the Data Buffer, which is limited in size as described in 18.

Table 19 Data Buffer for Random Chunk Operations continued

Element	Length (Bytes)	Description
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 65,535; however, the chunk descriptor must fit in the Data Buffer, which is limited in size as described in 18. • 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 transactional interface 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 transactional interface does not check for violations of this rule and does not properly handle such violations.

The following table shows a sample Data Buffer for a 32-bit application for fetching direct random chunks.

Element	Sample Value	Length (in 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

Element	Sample Value	Length (in Bytes)
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.

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

Table 20 Data Buffer for Rectangle Chunks

Element	Length (Bytes)	Description
Record Address	4	The 4-byte physical location of the record. You can retrieve this location with a Get Position operation (22).
Rectangle Chunk Descriptor		
Subfunction	4	Type of chunk descriptor; one of the following: • 0x80000002 (Direct rectangle chunk descriptor)—Retrieves chunks directly into the Data Buffer. The first chunk is retrieved and stored at offset 0 in the Data Buffer, the second chunk immediately follows the first, and so on. • 0x80000003 (Indirect rectangle chunk descriptor)—Retrieves chunks into addresses specified by the User Data and Application Distance Between Rows elements.
Number of Rows	4	Number of chunks on which the rectangle chunk descriptor must operate. The minimum value is 1. No explicit maximum value exists.
Offset	4	Offset from the beginning of the record of the first byte to retrieve. The minimum value is 0, and the maximum value is the offset of the last byte in the record. If the record is viewed as a rectangle, this element refers to the offset of the first byte in the first row to be retrieved.
Bytes Per Row	4	Number of bytes to retrieve in each chunk. The minimum value is 0, and the maximum value is 65,535; however, the chunk descriptor must fit in the Data Buffer, which is limited in size as described in 18.
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 location into which the transactional interface stores bytes after retrieving them from each row. For 64-bit applications, a 64-bit pointer to the location into which the transactional interface stores bytes after retrieving them from each row. The format you should use depends on your operating system.[1] The transactional interface ignores this element for direct rectangle descriptors; however, you must still allocate the element and initialize it to 0.

Table 20 Data Buffer for Rectangle Chunks continued

Element	Length (Bytes)	Description
Application Distance Between Rows	4	(Used only with indirect rectangle descriptors.) Number of bytes between the beginning of each chunk in the rectangle, as the rectangle is stored in your application's memory, at the address specified by User Data. The transactional interface 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.		

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 transactional interface uses the descriptor when copying the returned chunks to the locations that the User Data elements specify. In the event that you overwrite your chunk descriptor, the transactional interface returns Status Code 62.

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

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

The following table shows a sample Data Buffer for a 32-bit application for fetching a direct rectangle chunk.

Element Name	Sample Value	Length (in Bytes)
Record Address	0x00000628	4
Subfunction	0x80000002	4

Element Name	Sample Value	Length (in Bytes)
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 transactional interface calculates the subfunction's Offset element values based on your physical intrarecord currency (that is, your current physical location within the record). When you use the Next-in-Record subfunction, the transactional interface ignores the Offset element in the chunk descriptor.

Result

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

The transactional interface 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 transactional interface returning Status Code 103), or if any chunk's offset and length combine to exceed the length of the record. In the

latter case, the transactional interface 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 transactional interface returns one of these status codes, your application should check the Data Buffer Length parameter's return value to see how much data the transactional interface 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 transactional interface 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 transactional interface 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 the Get Position operation (22) to retrieve the 4-byte location of a record, save the location, and use Get Direct/Record to return directly to that location after performing other operations that affect currency.
- Use the 4-byte location to retrieve a record in a chain of duplicates without rereading all the records from the beginning of the chain.
- Change the current key path. A Get Position operation, followed by a Get Direct/Record operation with a different key number, establishes positioning for the current record in a different index path. A subsequent Get Next returns the next record in the file based on the new key path.

Parameters

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

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

Prerequisites

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

Procedure

1. Set the Operation Code to 23. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.
3. Store the 4-byte position of the requested record in the first 4 bytes of the Data Buffer.
4. Set the Data Buffer Length to a value greater than or equal to the length of the record to retrieve.
5. Set the Key Number to the key number of the path for which you want the transactional interface to establish logical currency. Specify -1 if you do not want the transactional interface to establish logical currency. To use the system-defined log key (also called system data), specify 125.

Result

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

If the Get Direct/Record operation is unsuccessful and the transactional interface cannot return the requested record, the transactional interface 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 transactional interface 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	✓					✓
Returned					✓	

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 transactional interface 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	✓	✓		✓	✓	✓
Returned		✓	✓	✓		

Prerequisites

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

Procedure

1. Set the Operation Code to 5. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.
3. Set the Data Buffer Length to a value greater than or equal to the length of the record to retrieve.
4. Specify the desired key value in the Key Buffer. 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.

Result

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

If the Get Equal operation is unsuccessful, the transactional interface 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 non-zero 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	✓	✓		✓		✓
Returned		✓	✓	✓	✓	

Prerequisites

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

Procedure

1. Set the Operation Code to 12. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.
3. Set the Data Buffer Length to a value greater than or equal to the length of the record to retrieve.
4. Indicate the Key Number for the key path. To use the system-defined log key (also called system data), specify 125.

Result

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

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

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

Positioning

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

Get Greater (8)

The Get Greater operation (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 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	✓	✓		✓	✓	✓
Returned		✓	✓	✓	✓	

Prerequisites

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

Procedure

1. Set the Operation Code to 8. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.
3. Set the Data Buffer Length to a value greater than or equal to the length of the record you want to retrieve.
4. Specify the desired key value in the Key Buffer parameter.

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

Result

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

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

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

Positioning

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

Get Greater Than or Equal (9)

The Get Greater Than or Equal operation (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 transactional interface 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	✓	✓		✓	✓	✓
Returned		✓	✓	✓	✓	

Prerequisites

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

Procedure

1. Set the Operation Code to 9. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.

3. Set the Data Buffer Length to a value greater than or equal to the length of the record you want to retrieve.
4. Specify the key value in the Key Buffer parameter.
5. Set the Key Number parameter to correspond to the correct key path. To use the system-defined log key (also called system data), specify 125.

Result

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

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

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

Positioning

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

Get Key (+50)

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

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

Parameters

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

Prerequisites

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

Procedure

1. Set the parameters as you would for the corresponding Get operation. You do not need to initialize the Data Buffer Length.
2. Set the Operation Code to the Get operation you want to perform, plus 50. For example, to perform a Get Key (+50) with the Get Equal operation (5), set the Operation code to 55.

The transactional interface does not allow Delete or Update operations after a Get Key operation (+50). Before the transactional interface 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 transactional interface 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 transactional interface cannot perform its passive concurrency conflict checking without the compare. When the Update or Delete fails, the transactional interface returns Status Code 8.

Result

If the transactional interface finds the requested key, it returns the key value in the Key Buffer and Status Code 0. Otherwise, the transactional interface 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 transactional interface 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 transactional interface 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	✓	✓		✓		✓
Returned		✓	✓	✓	✓	

Prerequisites

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

Procedure

1. Set the Operation Code to 13. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.
3. Set the Data Buffer Length to a value greater than or equal to the length of the record you want to retrieve.
4. Specify the Key Number for the key path. To use the system-defined log key (also called system data), specify 125.

Result

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

If the Get Last operation is unsuccessful, the transactional interface 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	✓	✓		✓	✓	✓
Returned		✓	✓	✓	✓	

Prerequisites

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

Procedure

1. Set the Operation Code to 10. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.
3. Set the Data Buffer Length to a value greater than or equal to the length of the record you want to retrieve.

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

Result

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

If the Get Less Than operation is unsuccessful, the transactional interface 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 transactional interface 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	✓	✓		✓	✓	✓
Returned		✓	✓	✓	✓	

Prerequisites

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

Procedure

1. Set the Operation Code to 11. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.

3. Set the Data Buffer Length to a value greater than or equal to the length of the record you want to retrieve.
4. Specify the key value in the Key Buffer parameter.
5. Set the Key Number parameter to the key path. To use the system-defined log key (also called system data), specify 125.

Result

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

If the Get Less Than or Equal operation is unsuccessful, the transactional interface 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	✓	✓		✓	✓	✓
Returned		✓	✓	✓	✓	

Prerequisites

- The file must be open.
- The file cannot be a data-only file.
- Your application must have an established logical next position based on the specified key.

Procedure

1. Set the Operation Code to 6. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.
3. Set the Data Buffer Length to a value greater than or equal to the length of the record you want to retrieve.
4. Specify the key value from the previous operation in the Key Buffer.

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

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

Result

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

If the Get Next operation is unsuccessful, the transactional interface 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 transactional interface lost positioning.

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

Positioning

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

Get Next Extended (36)

The Get Next Extended operation (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 checks to see if the examined record or records satisfy a filtering condition, and it retrieves the ones that do. The filtering condition is a logic expression and is not limited to key fields only.

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

Parameters

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

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 non-extended 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 more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.

3 Specify a large enough Data Buffer to accommodate either the input Data Buffer or the returned Data Buffer, whichever is larger. Initialize the Data Buffer according to the structure shown in Table 21.

4 Specify the Data Buffer Length as either the length of the input structure (Table 21) or the length of the returned structure (Table 22), whichever is larger.

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

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

Details

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

Table 21 Input Data Buffer Structure for Extended Get and Step Operations

Element	Length (Bytes)	Description
Header	2	Total length of the 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. For Step Next Extended operations, always set this value to “EG”.
Filter (Fixed Portion)	2	Maximum Reject Count, which is the number of records that the transactional interface can skip while searching for records that satisfy the filter condition. You can set the value from 0 to 65,535. (0 means the transactional interface uses the system-defined maximum reject count, which is 4,095.)
	2	Number of Terms in the logic expression of the filter condition. (0 means the transactional interface 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.

Table 21 Input Data Buffer Structure for Extended Get and Step Operations

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 in 11.
	2	Field Length.
	2	Field Offset (zero relative).
	1	Specifies a Comparison Code: 1 Equal 2 Greater than 3 Less than 4 Not equal 5 Greater than or equal 6 Less than or equal Add a +8 bias to compare strings using one of the file's existing ACSs. Add a +32 bias to compare strings using the file's default ACS, which is the first ACS defined in the file. If you use both a +8 bias and a +32 bias, the +32 bias is ignored.
		Add a +64 bias if the second operand is another field of the record, rather than a constant. Add a +128 bias to compare strings without case sensitivity.
	1	Indicates an AND/OR logic expression: 0—Identifies the last term 1—Next term is connected with AND 2—Next term is connected with OR
	2 or *n*	When comparing two fields: a 2-byte, zero-relative offset to the second field. (The second field must be the same type and length.) *or* When comparing a field to a constant: the actual value of the constant. The length of the constant (*n*) must equal the length of the field.
	0, 5, 9, or 17	When specifying an ACS by name (bias +8), the ACS identifier using one of the name formats shown in 12.

Table 21 Input Data Buffer Structure for Extended Get and Step Operations

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.
	2	Number of Fields to extract from each record.
Descriptor (Repeat this segment for each extracted field)	2	Field Length to extract.
	2	Field Offset (zero relative).

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

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

- The transactional interface finds the requested number of records that satisfy the filter.

- While the transactional interface 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 transactional interface reaches a rejected record after which no records can satisfy the filtering condition in the rest of the file.
- The transactional interface reaches the end of the file.

Examples

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

1. Set the Number of Records to 1.
2. Set the Number of Fields to 1.
3. Set the Field Length to the length of the entire record to retrieve.
4. Set the Field Offset to 0.

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

1. Set the Number of Records to 12.
2. Set the Number of Fields to 4.
3. Set the Field Length and Field Offset parameters for each of the 4 fields extracted.

Retrieving Fields from Records

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

Table 22 illustrates the structure of the Data Buffer that the transactional interface returns.

Table 22 Returned Data Buffer Structure for Extended Get and Step Ops

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

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

The transactional interface 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 transactional interface starts the field description for the next returned field immediately following the data for the current field.

For example, suppose your fixed-record length is 100 bytes, your variable-length portion is up to 300 bytes, and you want to return just the variable-length portion of 5 records. You would use the descriptor element of the input buffer to set a Field Length of 300

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

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

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

Result

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

- In the Data Buffer, one or more fields from one or more records. (See Table 22.)
- In the Data Buffer Length, the total number of bytes received.
- In the Key Buffer, the key value for the last data record received.

If the Get Next Extended operation is unsuccessful, the transactional interface 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 transactional interface lost positioning.
134	The transactional interface cannot read the International Sorting Rule.
135	The specified International Sort Rule table is corrupt or otherwise invalid.
136	The transactional interface cannot find the specified Alternate Collating Sequence in the file.

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

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

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

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

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

Depending on the fields and the operators used in the filtering condition, the transactional interface may be able to optimize your request. After reaching a certain rejected record, it returns Status

Code 64, indicating that no records can satisfy the filtering conditions in the rest of the file.

Positioning

The Get Next Extended operation establishes the complete logical and physical currencies. The last record examined becomes the current record. This record can be either a record that satisfies the filtering condition and is retrieved, or a record that does not satisfy the filtering condition and is rejected, but is still not past the optimization limit. For example, if the extended operation returns status 9, the current record is that last record in the file. If status 60 (reject count reached) is retuned, then the current record is the last record rejected. If status 64 (Filter Limit Reached) is returned, then the current record is the last record that satisfies the optimization criteria. Even thought the transactional interface 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 did satisfy that criteria.

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

Get Position (22)

The Get Position operation (B_GET_POSITION) returns the physical 4-byte position of the current record. Get Position fails if there is no established physical currency when you issue the operation. Once you determine a record's position (address), you can use the Get Direct/Record operation (23) to retrieve that record directly by its physical location in the file. The transactional interface does not perform any disk I/O to process a Get Position request.

Parameter

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

Prerequisites

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

Procedure

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

Result

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

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

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

Positioning

The Get Position operation has no effect on positioning.

Get Previous (7)

The Get Previous operation (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	✓	✓		✓	✓	✓
Returned		✓	✓	✓	✓	

Prerequisites

- The file must be open.
- The file cannot be a data-only file.
- Your application must have established a logical previous position based on the specified key.

Procedure

1. Set the Operation Code to 7. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.
3. Set the Data Buffer Length to a value greater than or equal to the length of the record you want to retrieve.

4. Specify the key value from the previous operation in the Key Buffer. Pass the Key Buffer exactly as the transactional interface returned it on the previous call. The transactional interface may need the information previously stored in the Key Buffer to determine its current position in the file.
5. Set the Key Number parameter to the key path used on the previous call. You cannot change key paths using a Get Previous operation.

Result

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

If the Get Previous operation is unsuccessful, the transactional interface 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 transactional interface lost positioning

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

Positioning

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

Get Previous Extended (37)

The Get Previous Extended operation (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 checks to see if the examined record or records satisfy a filtering condition, and it retrieves the ones that do. The filtering condition is a logic expression and is not limited to key fields only.

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

Parameters

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

Prerequisites

- The file must be open.
- The file cannot be a data-only file.
- You must have an established logical previous position based on the specified key.

Procedure

1. Set the Operation Code to 37. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.

3. Specify a large enough Data Buffer to accommodate either the input Data Buffer or the returned Data Buffer, whichever is larger. Initialize the Data Buffer according to the structure shown in Table 21.

4. Specify the Data Buffer Length as either the length of the input structure (Table 21) or the length of the returned structure (Table 22), whichever is larger.

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

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

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

Details

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

Result

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

Positioning

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

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

Insert (2)

The Insert operation (B_INSERT) inserts a record into a file. The transactional interface 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	✓	✓	✓	✓		✓
Returned		✓			✓	

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

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.

Result

If the Insert operation is successful, the transactional interface 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 transactional interface also

returns the inserted record in the Data Buffer, including the AUTOINCREMENT value assigned by the transactional interface. An NCC Insert operation does not change the value of the Key Buffer parameter.

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

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

Positioning

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

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

Note The transactional interface 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 transactional interface 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 transactional interface 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	✓	✓	✓	✓		✓
Returned		✓	✓		✓	

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 23.
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 transactional interface uses to establish currency. To use the NCC option, specify -1 (0xFF) for the Key Number. To use the system-defined log key (also called system data), specify 125.

Details

The following table shows the data buffer structure.

Table 23 Data Buffer Structure for the Insert Extended Operation

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

Result

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

If the operation is only partially successful and the transactional interface returns a nonzero status code, the first word of the Data Buffer equals the number of records that were successfully inserted. The record that caused the error is the number of records that were successfully inserted plus one.

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

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

Positioning

An Insert Extended operation that does not specify the NCC option establishes the complete logical and physical currencies and makes

the last inserted record the current one (unless the inserted record's key value is null). The logical currency is based on the specified key.

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

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

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

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

1. Get Position (22)—Obtains the 4-byte physical address for the logical current record. The application saves this value and passes it back in Step 3.
2. Insert Extended (40)—Inserts the new records. This operation establishes new logical and physical currencies.

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

The NCC Insert Extended operation has the same effect in terms of logical currency, but can have a different effect in terms of physical currency. For example, executing a Get Next (6) operation after either procedure produces the same result, but executing a Step Next (24) might return different records.

Login/Logout (78)

The Login/Logout operation (B_LOGIN/B_LOGOUT) allows a user to specify his/her user credentials and obtain authentication and authorization tokens from the database engine. This operation also allows the user to reset his/her login 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	✓				✓	✓
Returned						

Prerequisites

- The database name and the user ID must be pre-defined.

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 *Pervasive PSQL 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 *Pervasive PSQL Programmer's Guide*, see Database URIs.)

Result

If the Login or Logout operation is successful, 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 login and logout on every file. Instead, login once to a database at the beginning of a session, then logout 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 pathname.

Parameters

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

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 pathname of the file to open in the Key Buffer parameter. Terminate the pathname with a NULL (binary zero) depending on the setting for embedded spaces. The pathname 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 transactional interface normally expands the file name to a fully-qualified UNC file name. For example, J:\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 *Pervasive PSQL 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 name and share name. Even though you may get by with a longer path name if opened locally, remote clients may not be able to open the file.

Support for file names with embedded spaces based is enabled by a client configuration option, "Embedded Spaces." By default, this configuration parameter is set to 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 filenames that contain embedded spaces (such as "`C:\My Folder\my file.mkd`"). See *Advanced Operations Guide* (Long File Names and Embedded Spaces Support).

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

4 In the Key Number parameter, specify one of the mode values listed in Table 24.

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, because 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 transactional interface through the open mode 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.

Table 24 Open Modes

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 transactional interface accepted Accelerated mode opens, but interpreted them as Normal opens.) When you open a file in Accelerated mode, the transactional interface 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 transactional interface cannot automatically recover. If the data in the file's indexes has been corrupted, you can retrieve the records by opening the file in Read-Only mode and then using the Step Next (24) operation.	-2	8	101
Verify This mode is ignored. If you specify this mode, the transactional interface opens the file in Normal mode. In previous versions of the transactional interface, Verify mode verified that the data written to disk was correct.	-3	N/A	N/A
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 transactional interface. (The transactional interface recognizes and handles the situation in which more than one client at a time opens a file, or a single client has more than one Position Block in the file.) When you open an extended file, the transactional interface uses a single handle, and opens the base file and all extension files.

Result

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

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

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

The following tables show the possible combinations for open modes involving local clients.

Table 25 shows open modes involving local clients.

Table 25 Open Mode Combinations for Local Clients

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

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	✓				✓	✓
Returned						

Prerequisites

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

Procedure

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

Result

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

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

If the Reset operation is unsuccessful, the transactional interface 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 pathname.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	Key Buffer	Key Number
Sent	✓				✓	
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 transactional interface uses the default drive. If you do not specify the complete path for the directory, the transactional interface appends the directory path specified in the Key Buffer to the current directory.

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

Result

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

If the Set Directory operation is unsuccessful, the transactional interface 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 (a password) to a file. If an owner name has been set for a file, users or applications must specify the owner name each time they access the file. You can specify that an owner name be required for any access or just for update privileges.

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

Parameters

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

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 an owner name up to 24 bytes long (a "long" owner name). This bias is also defined in btrconst.h as B_LONG_OWNER_NAME_BIAS. Btrconst.h is provided with the Btrieve software development kit (SDK).

- Once a long owner name is specified, the data file cannot be read by a database engine prior to Pervasive PSQL v10.10.
- A data file with a long owner name cannot be rebuilt to a file format prior to 9.5 unless the owner name is first removed.

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 transactional interface requires that the name be in both buffers to avoid accidentally specifying an incorrect value.

   If the +17000 bias is not specified, the owner name can be up to 8 bytes long and must end with a binary 0. This is referred to as a "short" owner name. If the +17000 bias is specified, the owner name can be up to 24 bytes long and must end with a binary 0. This is referred to as a "long" owner name. In either case, the owner name cannot consist of all spaces (0x20).

4 Specify the length of the owner name, including the binary 0, in the Data Buffer Length parameter. If the owner name is shorter than the maximum length allowed (either 8 or 24 bytes), the owner name is padded with spaces.

5 Set the Key Number to an integer that specifies the type of access restrictions and encryption for the file. (See Table 26.)

Details

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

Table 26 Access and Encryption Codes

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

Result

If the Set Owner operation is successful, the transactional interface prevents future operations from accessing or modifying the file unless those operations specify the correct owner name. The only exception is if read-only access is allowed without an owner name. In addition, if the Set Owner operation is successful, the transactional interface encrypts the data in the file (if encryption is specified).

Encryption occurs immediately; the transactional interface has control until the entire file is encrypted, and the larger the file, the longer the encryption process takes. Reading data from an encrypted file is slower than reading data from an unencrypted file. The

transactional interface decrypts a page when it loads the page from the disk, then encrypts the page when it writes to the disk again. If you have a small cache or use a relatively large amount of modification operations, the transactional interface must execute the encryption routine more frequently.

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

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

Positioning

The Set Owner operation has no effect on positioning.

Stat (15)

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

Parameters

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

Prerequisites

The file must be open.

Procedure

1. Set the Operation Code to 15.
2. Pass the Position Block for the file.
3. Indicate a Data Buffer to hold the statistics defined for the file.
4. Specify the Data Buffer Length, which must be long enough to hold the file statistics. (For more information, see Table 27 and Table 28.)
5. Specify a Key Buffer that is at least 255 characters long.
6. Set the Key Number as follows:
   - Specify 0 to exclude file version and unused duplicate pointers information. Parse the returned Data Buffer as shown in Table 27.
   - Specify -1 (0xFF) to include file version and unused duplicate pointers information. Parse the returned Data Buffer as shown in Table 28.

Details

The transactional interface returns information about all keys in the file, including those added since file creation. The key information includes any applicable ACS definitions. You must account for this

extra information in the Data Buffer. Specifically, do not use the same Data Buffer here that you used for the Create (0) operation.

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

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

Table 27 Data Buffer Excluding File Version Information

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

Table 27 Data Buffer Excluding File Version Information continued

Element	Description	Length (Bytes)
	Key Number	1
	ACS Number	1
ACS Number 0	ACS	265
. . .	. . .	. . .
ACS Number *x*	ACS	265

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

Table 28 Data Buffer Including File Version Information

Element	Description	Length (Bytes)
File Specification	Record Length	2
	Page Size	2
	Number of Indexes	1
	File Version Number	1
	Number of Records	4
	File Flags	2
	Number of Unused Duplicate Pointers	1
	Reserved Byte	1
	Unused Pages	2
Key Specifications (repeated for each key segment)	Key Position	2
	Key Length	2

Table 28 Data Buffer Including File Version Information continued

Element	Description	Length (Bytes)
	Key Flags	2
	Number of Unique Key Values	4
	Extended Data Type	1
	Null Value	1
	Reserved	2
	Key Number	1
(last member of Key Specification, end of repeated elements)	ACS Number	1
ACS Number 0	ACS	265
. . .	. . .	. . .
ACS Number *x*	ACS	265

File Specifications

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

- In the File Specification area:
  - If the Data Buffer includes file version information, the Number of Indexes is 1 byte long and is followed by a 1-byte File Version Number. Do not translate the File Version Number value to decimal. A value of 0x70 indicates that the file is a 7.0 file; a value of 0x60 indicates that the file is 6.*x*, and so on. When creating a file, the transactional interface assigns a version number according to the attributes defined for the file.
  - The Number of Records is a 4-byte value representing the number of records in the file.
  - In the File Flags word, Bits 9 and 12 have the following meaning:

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

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

- If the Data Buffer includes file version information, a 1-byte Number of Unused Duplicate Pointers follows the File Flags field and indicates how many unused duplicate pointers remain in the file.
- The reserved areas are allocated even though the transactional interface 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 10, except that a 4-byte Number of Unique Key Values follows the Key Flags field and indicates the number of records that have a unique, non-duplicated value for the specified key.

ACSs

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

Result

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

If the Stat operation is unsuccessful, the transactional interface 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.

Table 29 Stat Extended (65) Subfunctions

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)

Parameters

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

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 extended stat structure in the Data Buffer. See the subsections below for more information about the extended stat 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 follows.

Table 30 Extended Files Descriptor

Element	Length (Bytes)	Description
Signature	4	Type of extended stat call. Specify the following 4 bytes to indicate an Extended Stat 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 extended stat call. Specify 0x00000001.
Namespace	4	File naming convention. Specify 0x00000000.
Max Files	4	Maximum number of filenames to return. You can set this value higher than the number of extension files composing the extended file. (An extended file can contain up to 32 extension files.)
First File Sequence	4	Sequence number of the first filename to return. Specify 0 to begin with the base file, 1 to begin with the first extension file, and so on. If you specify a number higher than the number of extension files, the transactional interface returns Status Code 0 and no filenames.
Buffer space	n	Allow enough additional room for the return data. If you receive Status Code 22, re-try the operation with a larger data buffer size.

Output Data Buffer Structure

For the extended files subfunction, the transactional interface updates the value of the Data Buffer Length parameter and returns

an extended files structure in the Data Buffer, as illustrated in Table 31 on page 2-141.

Table 31 Extended Files Return Buffer

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.
Filename Portion (Repeated for each filename returned)		
Length of Filename	4	Length of the extension filename.
Filename	*n*	Extension filename.

Subfunction 2: System Data Information

For the file specified by the input Position Block, this subfunction returns information about whether there is a system key defined on a file, and whether the file can be logged (transaction durable).

Input Data Buffer Structure

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

Table 32 System Data Descriptor

Element	Length (Bytes)	Description
Signature	4	Unique identifier for an extended stat call. Specify *ExSt*. See Table 30.
Subfunction	4	Type of extended stat call. Specify 0x00000002.

Output Data Buffer Structure

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

Table 33 System Data Return Buffer

Element	Length (Bytes)	Description
Has System Data	1	Indicates whether the file's records contain a system-defined log key (also called system data). 1 = Yes and 0 = No.
Has Log Key	1	Indicates whether the system-defined log key is currently being used, as opposed to being dropped. 1 = Yes and 0 = No (dropped).
Is Loggable	1	Indicates whether the file has a unique key that can be used to implement transaction durability. This key can be either a user-defined unique key or a system-defined log key. 1 = Yes and 0 = No.
Log Key Number	1	Key number that the transactional interface is currently using as the transaction log key. If the system-defined log key is being used as the transaction log key, this value is 125.
Size of System Data	2	Size of the system-defined log key, which is 8.
System Data Version	2	A two-byte field that contains the major version of the database engine. For example, 0x0009 for version 9.x and 0x000a for version 10.x. .

Subfunction 3: Duplicate Record Conflict 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 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), you must

create a duplicate record information descriptor in the Data Buffer, as follows.

Table 34 Duplicate Record Conflict Descriptor

Element	Length (Bytes)	Description
Signature	4	Unique identifier for an extended stat call. Specify *ExSt*. See Table 30.
Subfunction	4	Type of extended stat call. Specify 0x00000003.

Output Data Buffer Structure

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

Table 35 Duplicate Record Conflict Return Buffer

Element	Length (Bytes)	Description
Duplicate Record Address	4	Physical address of record containing the duplicate key value.
Key Number	4	Key number of the key containing the duplicate value.

Subfunction 4: 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: the internal file ID used by the transactional interface 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.

Table 36 File Information Descriptor - Open File

Element	Length (Bytes)	Description
Signature	4	Unique identifier for an extended stat call. Specify *ExSt*. See Table 30.
Subfunction	4	Type of extended stat call. Specify 0x00000004.
Buffer space	12	Additional bytes needed for returned information. See Output Data Buffer Structure. For the file information subfunction, the transactional interface returns a file information structure in the Data Buffer.

Output Data Buffer Structure

For the file information subfunction, the transactional interface returns a file information structure in the Data Buffer, as follows.

Table 37 File Information Structure - Open File

Element	Length (Bytes)	Description
FileID	4	A unique number which the transactional interface uses to identify the file.
NumberOfHandles	4	The current number of handles that the transactional interface has open on this file.
OpenTimeStamp	4	The system time when the physical file was last opened by the transactional interface. This system time is expressed as the number of seconds since midnight on January 1, 1970 in coordinated universal time (UTC) time.

Table 37 File Information Structure - Open File

Element	Length (Bytes)	Description
FileUsageCount	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 transactional interface cache. When a checkpoint starts, this number increments.
Flags	4	A four-byte bitmap in which various values may be set. See the table below 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 tables below.

Table 38 File Information Flags

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 transactional interface as Read Only. This may be a CD-ROM drive or a read-only directory.
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.

Table 38 File Information Flags

Value	Name	Description
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.

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.

Table 39 Gateway Information Descriptor

Element	Length (Bytes)	Description
Signature	4	Unique identifier for an extended stat call. Specify *ExSt*. See Table 30.
Subfunction	4	Type of extended stat 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 transactional interface modifies the Data Buffer Length parameter and returns a file information structure in the Data Buffer, as follows.

Table 40 File Information Structure - Gateway Information

Element	Length (Bytes)	Description
Major Version	4	The major version of the engine, such as version 7 or 8.
Minor Version	4	The minor version of the engine, such as 05 or 82.
Patch Level	4	The patch level of the engine, such as 1, 2, or 3.
Platform	4	The operating system platform the engine is running on.
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.

Table 41 Lock Owner Information Descriptor

Element	Length (Bytes)	Description
Signature	4	Unique identifier for an extended stat call. Specify *ExSt*. See Table 30.
Subfunction	4	Type of extended stat 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 transactional interface modifies the Data Buffer Length parameter and returns a file information structure in the Data Buffer, as follows.

Table 42 Lock Owner Information Return Buffer

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 table below 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 non-zero, 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 Pervasive Software.
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 transactional interface 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 table below.

Table 43 Lock Owner Flags

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 computer. 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.
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 what permissions it has, you must create a security information descriptor in the Data Buffer, as follows.

Table 44 Security Information Descriptor

Element	Length (in Bytes)	Description
Signature	4	Type of extended stat call. Specify 0x45785374. (This is equivalent to ASCII ExSt.)
Signature	4	Type of extended stat 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.

Table 45 Security Information Return Buffer

Element	Length (in Bytes)	Description
Flags for Handle	4	A four-byte bitmap containing flags indicating the methods used for security on this handle. See the table below for a description of each flag value.
Flags for Handle	4	A four-byte bitmap containing flags indicating the methods used for security on the current default database for this client. See the table below for a description of each flag value.
Permissions	4	These are the permissions granted to the client using this handle. See the permission table below for a description of each flag value.

Table 45 Security Information Return Buffer

Element	Length (in Bytes)	Description
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 *Pervasive PSQL 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.
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 tables below.

Table 46 Security Flags

Value	Name	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 the Linux Operating System, this indicates that etc/btpasswd file was used.
0x00000100	PAM	If Authentication is by the Linux 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.

Table 47 Permission Flags

Value	Name	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.

Result

If the Stat Extended operation is unsuccessful, the transactional interface 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 transactional interface 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	✓	✓		✓		
Returned		✓	✓	✓		

Prerequisites

The file must be open.

Procedure

1. Set the Operation Code to 33. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.
3. Set the Data Buffer Length to a value greater than or equal to the length of the record to retrieve.

Result

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

If the Step First operation is unsuccessful, the transactional interface 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 transactional interface 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	✓	✓		✓		
Returned		✓	✓	✓		

Prerequisites

The file must be open.

Procedure

1. Set the Operation Code to 34. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.
3. Set the Data Buffer Length to a value greater than or equal to the length of the record to retrieve.

Result

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

If the Step Last operation is unsuccessful, the transactional interface 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 transactional interface 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	✓	✓		✓		
Returned		✓	✓	✓		

Prerequisites

The file must be open.

Procedure

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

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.
3. Set the Data Buffer Length to a value greater than or equal to the length of the record to retrieve.

Result

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

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

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

Positioning

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

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

If a Step Next operation is issued immediately after an Open operation (0), Step Next returns the first record in the file.

Step Next Extended (38)

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

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

Parameters

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

Prerequisites

- The file must be open.
- You must have established a next physical position. (For example, a Step Next Extended operation cannot follow a Delete operation.)

Procedure

1. Set the Operation Code to 38. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.

3 Specify a Data Buffer large enough to accommodate either the input data buffer or the returned data buffer, whichever is larger. Initialize the Data Buffer according to the structure shown in Table 21.

4 Specify the Data Buffer Length from the preceding step.

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

Details

The Step Next Extended operation shares the same "Details" as the Get Next Extended operation. Refer to Details for more information.

Result

If the Step Next Extended operation is successful, the transactional interface returns one or more fields from one or more records to the Data Buffer (as shown in Table 22). The transactional interface also sets the Data Buffer Length parameter to the number of bytes it returned to the Data Buffer.

If the Step Next Extended operation is unsuccessful, the transactional interface returns one of the following status codes:

3	The file is not open.
9	The operation encountered the end-of-file.
22	The data buffer parameter is too short.
60	The specified reject count has been reached.
61	The work space is too small.
62	The descriptor is incorrect.
64	The filter limit has been reached.
65	The field offset is incorrect.
82	The transactional interface lost positioning.
134	The transactional interface cannot read the International Sorting Rule.

135	The specified International Sort Rule table is corrupt or otherwise invalid.
136	The transactional interface cannot find the specified Alternate Collating Sequence in the file.

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

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

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

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

When the transactional interface 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 transactional interface finds that only 5 bytes are available (between offset 45 and the end of the record, at offset 49). At this point, the transactional interface does not pad the missing 5 bytes of field 2, and thus cannot extract field 3. Instead, the transactional interface returns Status Code 22 and places all of field 1 and first 5 bytes of field 2 in the return Data Buffer.

Positioning

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

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

Step Previous (35)

The Step Previous operation (B_STEP_PREVIOUS) retrieves the record to which the previous physical position points. The transactional interface 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	✓	✓		✓		
Returned		✓	✓	✓		

Prerequisites

- The file must be open.
- You must have an established previous physical position. (For example, a Step Previous cannot follow a Delete operation.)

Procedure

1. Set the Operation Code to 35. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.
3. Set the Data Buffer Length to a value greater than or equal to the length of the record to retrieve.

Result

If the operation is successful, the transactional interface returns the previous physical record in the Data Buffer and sets the Data Buffer Length parameter to the number of bytes returned.

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

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

Positioning

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

Step Previous Extended (39)

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

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

Parameters

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

Prerequisites

- The file must be open.
- You must have established a previous physical position. (For example, a Step Previous Extended operation cannot follow a Delete operation.)

Procedure

1. Set the Operation Code to 39. Optionally, you can include a lock bias:
   - +100—Single wait record lock.
   - +200—Single no-wait record lock.
   - +300—Multiple wait record lock.
   - +400—Multiple no-wait record lock.

   For more information about locking, refer to the *Pervasive PSQL Programmer's Guide*.
2. Pass the Position Block for the file.
3. Specify a Data Buffer large enough to accommodate either the input data buffer or the returned data buffer. Initialize the Data Buffer according to the structure shown in Table 21.

4 Specify the Data Buffer Length from the preceding step.

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

Details

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

Result

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

Positioning

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

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

Stop (25)

The Stop operation (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	✓					
Returned						

Procedure

Set the Operation Code to 25.

Result

If the Stop operation is successful, the transactional interface 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 transactional interface) exist and depending on the transactional interface configuration, the transactional interface may terminate itself and free a number of resources.

If the Stop operation is unsuccessful, the transactional interface returns a nonzero status code. The most common nonzero Status Code is 20 (Record Manager Inactive). This status occurs because the transactional interface 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	✓	✓	✓	✓		✓
Returned						

Prerequisites

You must have at least one record lock.

Procedure

➤ **To unlock a single-record lock, follow these steps:**

1. Set the Operation Code to 27.
2. Pass the Position Block for the file that contains the locked record.
3. Set the Key Number to a non-negative value.

➤ **To unlock a record locked by a multiple-record lock, first retrieve the 4-byte position of the record to unlock by issuing a Get Position operation (22) for that record. Then, issue the Unlock operation as follows:**

1. Set the Operation Code to 27.
2. Pass the Position Block for the file that contains the locked record.
3. Store (in the Data Buffer) the 4-byte position that the transactional interface returns.

4 Set the Data Buffer Length to 4.

5 Set the Key Number parameter to -1.

➢ **To unlock all the multiple record locks on a file, follow these steps:**

1 Set the Operation Code to 27.

2 Pass the Position Block for the file that contains the multiple locks.

3 Set the Key Number parameter to -2.

Result

If the Unlock operation is successful, the transactional interface releases all the locks that the operation specified.

If the Unlock operation is unsuccessful, the transactional interface 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	✓	✓	✓	✓		✓
Returned		✓			✓	

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

Prerequisites

- The file must be open.
- You must have established physical currency in the file. (Although an Extended Get, Extended Step, or Get Key operation establishes the required position, these operations cannot be followed by an Update.)

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 (0xFF) for the Key Number. To use the system-defined log key (also called system data), specify 125.

   When performing a non-NCC Update operation immediately following a Get operation, pass the Key Number exactly as the transactional interface returned it on the Get operation; otherwise, the transactional interface updates the record successfully but returns Status Code 7 on the first Get operation performed after the update.

Result

If the Update operation is successful, the transactional interface 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 transactional interface releases the lock. However, a multiple-record lock is never released by an Update operation.

If the Update operation is unsuccessful, the transactional interface 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 transactional interface 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 an INTEGER key's logical current record has a value of 1. For that same key, the logical next record has a value of 2. If you update 1 to 4, you no longer have the same logical next record. In this example, after the Update operation, the logical next record has a value that is greater than 4.

An NCC Update operation does not affect logical currency. This means that an application, having performed an NCC Update operation, has the same logical position in the file as it had prior to the Update operation. In such a situation, operations that follow an NCC Update—such as Get Next (6), Get Next Extended (36), Get Previous (7), and Get Previous Extended (37)—return values based on the application's logical currency prior to the NCC Update.

Note The transactional interface 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	✓	✓	✓	✓		✓
Returned		✓			✓	

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 operation (+50) establishes the required position, you cannot issue an Update Chunk operation immediately after these operations, because they do not return a single record.

Procedure

1. Set the Operation Code to 53.
2. Pass the Position Block for the file containing the record.
3. Specify a Data Buffer, as described in Details.
4. Set the Data Buffer Length to a value greater than or equal to the number of bytes your application has placed in the Data Buffer. See the “Details” section for more information about calculating the Data Buffer Length.
5. Set the Key Number used for retrieving the record in the Key Number parameter. To use the system-defined log key (also called system data), specify 125.

Details

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

- Random Chunk Descriptor—To update a single chunk per operation, or to update more than one chunk in a single operation when the chunks are spaced randomly throughout the record.
- Rectangle Chunk Descriptor—To update many chunks in an operation, when each chunk is the same length and chunks are spaced equidistantly in the record.
- Truncate Chunk Descriptor—To truncate a record at a specified offset.

Random Chunk Descriptor Structure

The following example shows a record with three randomly spaced chunks (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	0 A	0 B	0 C	0 D	0 E	0F
10	11	[*]	[*]	[*]	[*]	16	17	18	19	1 A	1 B	1 C	1 D	1 E	1F
20	21	22	23	24	25	26	27	28	29	[*]	[*]	[*]	[*]	[*]	[*]
[*]	[*]	32	33	34	35	36	37	38	39	3 A	3 B	3 C	3 D	3 E	3F
40	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	4F

To define a random chunk descriptor, your application must create a structure in the Data Buffer, based on the following table.

Table 48 Random Chunk Descriptor Structure

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.
NumChunks	4	Number of chunks to be updated. The value must be at least 1. Although no explicit maximum value exists, the chunk definitions must fit in the Data Buffer, which is limited in size as described in 18.
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 65,535; however, the chunk definitions must fit in the Data Buffer, which is limited in size as described in 18. • 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 transactional interface 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 transactional interface 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 (in Bytes)
Subfunction	0x8000000	4
NumChunks	3	4
Chunk 0		
Chunk Offset	0x12	4
Chunk Length	0x05	4
User Data	N/A	4
Chunk 1		
Chunk Offset	0x2A	4
Chunk Length	0x08	4
User Data	N/A	4
Chunk 2		
Chunk Offset	0x41	4
Chunk Length	0x0E	4
User Data	N/A	4
Data for Chunk 0	N/A	5
Data for Chunk 1	N/A	8
Data for Chunk 2	N/A	14

Rectangle Chunk Descriptor Structure

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

00	01	02	03	04	05	06	07	08	09	0 A	0 B	0 C	0 D	0 E	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 Table 49.

Table 49 Rectangle Chunk Descriptor Structure

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.

Table 49 Rectangle Chunk Descriptor Structure

Element	Length (Bytes)	Description
Bytes Per Row	4	Number of bytes in each chunk to be updated. The minimum value is 0, and the maximum value is 65,535; however, the chunk definitions must fit in the Data Buffer, which is limited in size as described in 18.
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 transactional interface 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 your application's memory, at the address specified by User Data. The transactional interface 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 your application's memory with either more or fewer bytes between rows, Application Distance Between Rows allows you to pass that information to the transactional interface.

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

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

Element Name	Sample Value	Length (in 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:

Table 50 Truncate Descriptor Structure

Element	Length (in Bytes)	Description
Subfunction	4	Type of chunk descriptor. Specify 0x80000004.
ChunkOffset	4	Byte offset into the record where truncation begins. That byte and every byte following it is eliminated. The minimum value is 4. The maximum value is the offset of the final byte in the record.

Next-in-Record Subfunction Bias

If you add a bias of 0x40000000 to any of the subfunctions previously listed, the transactional interface calculates the subfunction's Offset element values based on your physical intrarecord currency (that is, your current physical position within the record). When you use the

Next-in-Record subfunction, the transactional interface 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 transactional interface calculates the offset for all chunks (except the first) by adding the previous chunk's length to the previous chunk's offset. In other words, the next-in-record bias applies to all chunks in the operation.

Append Subfunction Bias

If you add a bias of 0x20000000 to the random chunk descriptor's subfunction or to the rectangle chunk descriptor's subfunction, the transactional interface calculates the subfunction's Offset element value to be one byte beyond the end of the record.

Note Do not use this bias with the Next-in-Record bias or the Truncate subfunction.

If you use this bias in combination with a random chunk descriptor and it updates more than one chunk in a single operation, the transactional interface calculates the offset for all chunks (except the first chunk) based on the record's length after the transactional interface appends the previous chunk.

Result

If the Update Chunk operation is successful, the transactional interface 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 transactional interface 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 transactional interface releases the lock. However, a multiple-record lock is never released by an Update Chunk operation.

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

Version (26)

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

Parameters

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

Prerequisites

Either the transactional interface or the Requester must be loaded before you can issue a Version operation.

Procedure

1. Set the Operation Code to 26.
2. Set the Data Buffer Length to at least 15. (For more information, see Table 51.)
3. To retrieve the version number of a server-based transactional interface, you must specify either a valid Position Block for an opened file on that server or a valid pathname in the Key Buffer.

Result

If you have both a workstation transactional interface and client Requester configured for access and the Version operation is successful, the operation returns the version information for the workstation transactional interface, the client Requester, and the server-based transactional interface.

Specify a 15-byte Data Buffer and Data Buffer Length.

If both the client Requester and the workstation transactional interface are loaded and you specify only a 5-byte Data Buffer and Data Buffer Length, the operation returns only the client Requester's version information.

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 transactional interface or Requester, according to the format shown in Table 51. The fifth byte of each block identifies each transactional interface or Requester.

Table 51 Version Block

Element	Length (Bytes)	Description
Version Number	2	Pervasive PSQL version number.
Revision Number	2	Pervasive PSQL revision number.
Requester or Engine Type	1	Type of engine or requester; one of the following: • 9 (0x39) for the Workgroup database engine or Linux database server using Workgroup authentication mode • D (0x44) for DOS workstation • N (0x4E) for client Requester • T (0x54) for Windows server engine • U (0x55) for Linux server using PAM or BTPASSWD authentication

For example, if you are running Pervasive.SQL v8.10 for Windows, the Version operation returns the following hexadecimal values in the Data Buffer:

```
08 00 0A 00 54
```

After converting these values to decimal, the version number is 8 and the revision number is 10. If the Version operation is unsuccessful, the transactional interface returns a nonzero status code.

Positioning

The Version operation has no effect on positioning.

Appendix

Quick Reference of Btrieve Operations

A

This appendix provides a summary of Btrieve API operations in numerical order by operation code.

Table of Btrieve API Operations

.

Table 52 Quick Reference of Btrieve API Operations

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.

Table 52 Quick Reference of Btrieve API Operations continued

Operation	Code	Constant	Description
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.
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 pathname.
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.

Table 52 Quick Reference of Btrieve API Operations continued

Operation	Code	Constant	Description
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 portions (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 workstation transactional interface (not available for server-based transactional interfaces).
Version	26	B_VERSION	Returns the version number of the transactional interface.
Unlock	27	B_UNLOCK	Unlocks a record or records.
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.

Table 52 Quick Reference of Btrieve API Operations continued

Operation	Code	Constant	Description
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 transactional interface files.
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 filenames and paths of an extended file's components and reports whether a file is using a system-defined log key.
Single-record wait lock	+100	S_WAIT_LOCK	Locks only one record at a time. If the record is already locked, the transactional interface 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 transactional interface 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 transactional interface retries the operation.

Table 52 Quick Reference of Btrieve API Operations continued

Operation	Code	Constant	Description
Multiple-record no-wait lock	+400	M_NOWAIT_LOCK	Locks several records concurrently in the same file. If the record is already locked, the transactional interface returns an error status code.
No-wait page lock	+500	NOWRITE_WAIT	In a concurrent transaction, tells the transactional interface not to wait if the page to be changed has already been changed by another active concurrent transaction. This bias can be combined with any of the record locking biases (+100, +200, +300, or +400).

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W