Oracle Berkeley DB

Installation and Build Guide

11g Release 2

(Library Version 11.2.5.1)

Legal Notice

Published 10/25/2011

Table of Contents

Preface

Welcome to Berkeley DB (DB). This document describes how to build, install and upgrade Berkeley DB

This document reflects Berkeley DB 11*g* Release 2, which provides DB library version 11.2.5.1.

Conventions Used in this Book

The following typographical conventions are used within in this manual:

Structure names are represented in `monospaced font`, as are `method names`. For example: "`DB->open()` is a method on a `DB` handle."

Variable or non-literal text is presented in *italics*. For example: "Go to your *DB_INSTALL* directory."

Program examples are displayed in a `monospaced font` on a shaded background. For example:

```
/* File: gettingstarted_common.h */
typedef struct stock_dbs {
    DB *inventory_dbp; /* Database containing inventory information */
    DB *vendor_dbp;    /* Database containing vendor information */

    char *db_home_dir;       /* Directory containing the database files */
    char *inventory_db_name; /* Name of the inventory database */
    char *vendor_db_name;    /* Name of the vendor database */
} STOCK_DBS;
```

Note

Finally, notes of interest are represented using a note block such as this.

For More Information

Beyond this manual, you may also find the following sources of information useful when building a DB application:

- Getting Started with Transaction Processing for C
- Berkeley DB Getting Started with Replicated Applications for C
- Berkeley DB C API
- Berkeley DB C++ API
- Berkeley DB STL API
- Berkeley DB TCL API

- Berkeley DB Programmer's Reference Guide
- Berkeley DB Upgrade Guide
- Berkeley DB Getting Started with the SQL APIs

To download the latest Berkeley DB documentation along with white papers and other collateral, visit http://www.oracle.com/technetwork/indexes/documentation/index.html.

For the latest version of the Oracle Berkeley DB downloads, visit http://www.oracle.com/technetwork/database/berkeleydb/downloads/index.html.

Contact Us

You can post your comments and questions at the Oracle Technology (OTN) forum for Oracle Berkeley DB at: http://forums.oracle.com/forums/forum.jspa?forumID=271, or for Oracle Berkeley DB High Availability at: http://forums.oracle.com/forums/forum.jspa?forumID=272.

For sales or support information, email to: berkeleydb-info_us@oracle.com You can subscribe to a low-volume email announcement list for the Berkeley DB product family by sending email to: bdb-join@oss.oracle.com

Chapter 1. Introduction

Welcome to the Berkeley DB. This manual describes how to configure, build and install Berkeley DB. Installation of DB for all of the platforms it officially supports is described in this manual. Upgrade instructions and release notes for newer versions of this product are described here. For infomation on upgrading from historical versions, see the Berkeley DB Upgrade Guide.

Note that some operating systems and distributions might provide DB, either by default or as part of an installation option. If so, those platforms will have installation instructions for DB specific to them. In this situation, you should see the documentation for your operating system or distribution provider for information on how to get DB on your platform.

Installation Overview

Berkeley DB is an open-source product, and as such it is usually offered in source-code format. This means that placing DB on your platform requires you to configure the build scripts, compile it, and then install the product onto your host system. The exception to this are Microsoft Windows platforms for which a binary installer is available. Note that for Windows platforms, you can still compile the product from source if you desire.

For *nix systems, including the BSD and Linux systems, the usual `configure`, `make` and `make install` installation process is used to place DB on your platform.

For information on building and installing Berkeley DB on:

- Microsoft Windows, see Building Berkeley DB for Windows (page 17) or Building Berkeley DB for Windows Mobile (page 32).
- Unix/POSIX — including Linux, BSD, iPhone OS, and Mac OS X — see Building Berkeley DB for UNIX/POSIX (page 37).
- VxWorks, see Building Berkeley DB for VxWorks (page 67).

Chapter 2. System Installation Notes

File utility /etc/magic information

The `file`(1) utility is a UNIX utility that examines and classifies files, based on information found in its database of file types, the /etc/magic file. The following information may be added to your system's /etc/magic file to enable `file`(1) to correctly identify Berkeley DB database files.

The `file`(1) utility `magic`(5) information for the standard System V UNIX implementation of the `file`(1) utility is included in the Berkeley DB distribution for both big-endian (for example, Sparc) and little-endian (for example, x86) architectures. See Big-endian magic information (page 3) and Little-endian magic information (page 6) respectively for this information.

The `file`(1) utility `magic`(5) information for Release 3.X of Ian Darwin's implementation of the file utility (as distributed by FreeBSD and most Linux distributions) is included in the Berkeley DB distribution. This `magic.txt` information is correct for both big-endian and little-endian architectures. See the next section for this information.

Magic information

```
# Berkeley DB
#
# Ian Darwin's file /etc/magic files: big/little-endian version.
#
# Hash 1.85/1.86 databases store metadata in network byte order.
# Btree 1.85/1.86 databases store the metadata in host byte order.
# Hash and Btree 2.X and later databases store the metadata in
# host byte order.

0 long 0x00061561 Berkeley DB
>8 belong 4321
>>4 belong >2  1.86
>>4 belong <3  1.85
>>4 belong >0  (Hash, version %d, native byte-order)
>8 belong 1234
>>4 belong >2  1.86
>>4 belong <3  1.85
>>4 belong >0  (Hash, version %d, little-endian)

0 belong 0x00061561 Berkeley DB
>8 belong 4321
>>4 belong >2  1.86
>>4 belong <3  1.85
>>4 belong >0  (Hash, version %d, big-endian)
>8 belong 1234
>>4 belong >2  1.86
>>4 belong <3  1.85
>>4 belong >0  (Hash, version %d, native byte-order)
```

```
0 long 0x00053162 Berkeley DB 1.85/1.86
>4 long >0  (Btree, version %d, native byte-order)
0 belong 0x00053162 Berkeley DB 1.85/1.86
>4 belong >0  (Btree, version %d, big-endian)
0 lelong 0x00053162 Berkeley DB 1.85/1.86
>4 lelong >0  (Btree, version %d, little-endian)

12 long 0x00061561 Berkeley DB
>16 long >0  (Hash, version %d, native byte-order)
12 belong 0x00061561 Berkeley DB
>16 belong >0  (Hash, version %d, big-endian)
12 lelong 0x00061561 Berkeley DB
>16 lelong >0  (Hash, version %d, little-endian)

12 long 0x00053162 Berkeley DB
>16 long >0  (Btree, version %d, native byte-order)
12 belong 0x00053162 Berkeley DB
>16 belong >0  (Btree, version %d, big-endian)
12 lelong 0x00053162 Berkeley DB
>16 lelong >0  (Btree, version %d, little-endian)

12 long 0x00042253 Berkeley DB
>16 long >0  (Queue, version %d, native byte-order)
12 belong 0x00042253 Berkeley DB
>16 belong >0  (Queue, version %d, big-endian)
12 lelong 0x00042253 Berkeley DB
>16 lelong >0  (Queue, version %d, little-endian)

12 long 0x00040988 Berkeley DB
>16 long >0  (Log, version %d, native byte-order)
12 belong 0x00040988 Berkeley DB
>16 belong >0  (Log, version %d, big-endian)
12 lelong 0x00040988 Berkeley DB
>16 lelong >0  (Log, version %d, little-endian)
```

Big-endian magic information

```
# Berkeley DB
#
# System V /etc/magic files: big-endian version.
#
# Hash 1.85/1.86 databases store metadata in network byte order.
# Btree 1.85/1.86 databases store the metadata in host byte order.
# Hash and Btree 2.X and later databases store the metadata in
# host byte order.

0 long 0x00053162 Berkeley DB 1.85/1.86 (Btree,
>4 long 0x00000002 version 2,
>4 long 0x00000003 version 3,
```

```
>0 long 0x00053162 native byte-order)

0 long 0x62310500 Berkeley DB 1.85/1.86 (Btree,
>4 long 0x02000000 version 2,
>4 long 0x03000000 version 3,
>0 long 0x62310500 little-endian)

12 long 0x00053162 Berkeley DB (Btree,
>16 long 0x00000004 version 4,
>16 long 0x00000005 version 5,
>16 long 0x00000006 version 6,
>16 long 0x00000007 version 7,
>16 long 0x00000008 version 8,
>16 long 0x00000009 version 9,
>12 long 0x00053162 native byte-order)

12 long 0x62310500 Berkeley DB (Btree,
>16 long 0x04000000 version 4,
>16 long 0x05000000 version 5,
>16 long 0x06000000 version 6,
>16 long 0x07000000 version 7,
>16 long 0x08000000 version 8,
>16 long 0x09000000 version 9,
>12 long 0x62310500 little-endian)

0 long 0x00061561 Berkeley DB
>4 long >2  1.86
>4 long <3  1.85
>0 long 0x00061561 (Hash,
>4 long 2  version 2,
>4 long 3  version 3,
>8 long 0x000004D2 little-endian)
>8 long 0x000010E1 native byte-order)

12 long 0x00061561 Berkeley DB (Hash,
>16 long 0x00000004 version 4,
>16 long 0x00000005 version 5,
>16 long 0x00000006 version 6,
>16 long 0x00000007 version 7,
>16 long 0x00000008 version 8,
>16 long 0x00000009 version 9,
>12 long 0x00061561 native byte-order)

12 long 0x61150600 Berkeley DB (Hash,
>16 long 0x04000000 version 4,
>16 long 0x05000000 version 5,
>16 long 0x06000000 version 6,
>16 long 0x07000000 version 7,
>16 long 0x08000000 version 8,
```

```
>16 long 0x09000000 version 9,
>12 long 0x61150600 little-endian)

12 long 0x00042253 Berkeley DB (Queue,
>16 long 0x00000001 version 1,
>16 long 0x00000002 version 2,
>16 long 0x00000003 version 3,
>16 long 0x00000004 version 4,
>16 long 0x00000005 version 5,
>16 long 0x00000006 version 6,
>16 long 0x00000007 version 7,
>16 long 0x00000008 version 8,
>16 long 0x00000009 version 9,
>12 long 0x00042253 native byte-order)

12 long 0x53220400 Berkeley DB (Queue,
>16 long 0x01000000 version 1,
>16 long 0x02000000 version 2,
>16 long 0x03000000 version 3,
>16 long 0x04000000 version 4,
>16 long 0x05000000 version 5,
>16 long 0x06000000 version 6,
>16 long 0x07000000 version 7,
>16 long 0x08000000 version 8,
>16 long 0x09000000 version 9,
>12 long 0x53220400 little-endian)

12 long 0x00040988 Berkeley DB (Log,
>16 long 0x00000001 version 1,
>16 long 0x00000002 version 2,
>16 long 0x00000003 version 3,
>16 long 0x00000004 version 4,
>16 long 0x00000005 version 5,
>16 long 0x00000006 version 6,
>16 long 0x00000007 version 7,
>16 long 0x00000008 version 8,
>16 long 0x00000009 version 9,
>16 long 0x0000000a version 10,
>16 long 0x0000000b version 11,
>16 long 0x0000000c version 12,
>16 long 0x0000000d version 13,
>16 long 0x0000000e version 14,
>16 long 0x0000000f version 15,
>12 long 0x00040988 native byte-order)

12 long 0x88090400 Berkeley DB (Log,
>16 long 0x01000000 version 1,
>16 long 0x02000000 version 2,
>16 long 0x03000000 version 3,
```

```
>16 long 0x04000000 version 4,
>16 long 0x05000000 version 5,
>16 long 0x06000000 version 6,
>16 long 0x07000000 version 7,
>16 long 0x08000000 version 8,
>16 long 0x09000000 version 9,
>16 long 0x0a000000 version 10,
>16 long 0x0b000000 version 11,
>16 long 0x0c000000 version 12,
>16 long 0x0d000000 version 13,
>16 long 0x0e000000 version 14,
>16 long 0x0f000000 version 15,
>12 long 0x88090400 little-endian)
```

Little-endian magic information

```
# Berkeley DB
#
# System V /etc/magic files: little-endian version.
#
# Hash 1.85/1.86 databases store metadata in network byte order.
# Btree 1.85/1.86 databases store the metadata in host byte order.
# Hash and Btree 2.X and later databases store the metadata in
# host byte order.

0 long 0x00053162 Berkeley DB 1.85/1.86 (Btree,
>4 long 0x00000002 version 2,
>4 long 0x00000003 version 3,
>0 long 0x00053162 native byte-order)

0 long 0x62310500 Berkeley DB 1.85/1.86 (Btree,
>4 long 0x02000000 version 2,
>4 long 0x03000000 version 3,
>0 long 0x62310500 big-endian)

12 long 0x00053162 Berkeley DB (Btree,
>16 long 0x00000004 version 4,
>16 long 0x00000005 version 5,
>16 long 0x00000006 version 6,
>16 long 0x00000007 version 7,
>16 long 0x00000008 version 8,
>16 long 0x00000009 version 9,
>12 long 0x00053162 native byte-order)

12 long 0x62310500 Berkeley DB (Btree,
>16 long 0x04000000 version 4,
>16 long 0x05000000 version 5,
>16 long 0x06000000 version 6,
>16 long 0x07000000 version 7,
>16 long 0x08000000 version 8,
```

```
>16 long 0x09000000 version 9,
>12 long 0x62310500 big-endian)

0 long 0x61150600 Berkeley DB
>4 long >0x02000000 1.86
>4 long <0x03000000 1.85
>0 long 0x00061561 (Hash,
>4 long 0x02000000 version 2,
>4 long 0x03000000 version 3,
>8 long 0xD2040000 native byte-order)
>8 long 0xE1100000 big-endian)

12 long 0x00061561 Berkeley DB (Hash,
>16 long 0x00000004 version 4,
>16 long 0x00000005 version 5,
>16 long 0x00000006 version 6,
>16 long 0x00000007 version 7,
>16 long 0x00000008 version 8,
>16 long 0x00000009 version 9,
>12 long 0x00061561 native byte-order)

12 long 0x61150600 Berkeley DB (Hash,
>16 long 0x04000000 version 4,
>16 long 0x05000000 version 5,
>16 long 0x06000000 version 6,
>16 long 0x07000000 version 7,
>16 long 0x08000000 version 8,
>16 long 0x09000000 version 9,
>12 long 0x61150600 big-endian)

12 long 0x00042253 Berkeley DB (Queue,
>16 long 0x00000001 version 1,
>16 long 0x00000002 version 2,
>16 long 0x00000003 version 3,
>16 long 0x00000004 version 4,
>16 long 0x00000005 version 5,
>16 long 0x00000006 version 6,
>16 long 0x00000007 version 7,
>16 long 0x00000008 version 8,
>16 long 0x00000009 version 9,
>12 long 0x00042253 native byte-order)

12 long 0x53220400 Berkeley DB (Queue,
>16 long 0x01000000 version 1,
>16 long 0x02000000 version 2,
>16 long 0x03000000 version 3,
>16 long 0x04000000 version 4,
>16 long 0x05000000 version 5,
>16 long 0x06000000 version 6,
```

```
>16 long 0x07000000 version 7,
>16 long 0x08000000 version 8,
>16 long 0x09000000 version 9,
>12 long 0x53220400 big-endian)

12 long 0x00040988 Berkeley DB (Log,
>16 long 0x00000001 version 1,
>16 long 0x00000002 version 2,
>16 long 0x00000003 version 3,
>16 long 0x00000004 version 4,
>16 long 0x00000005 version 5,
>16 long 0x00000006 version 6,
>16 long 0x00000007 version 7,
>16 long 0x00000008 version 8,
>16 long 0x00000009 version 9,
>16 long 0x0000000a version 10,
>16 long 0x0000000b version 11,
>16 long 0x0000000c version 12,
>16 long 0x0000000d version 13,
>16 long 0x0000000e version 14,
>16 long 0x0000000f version 15,
>12 long 0x00040988 native byte-order)

12 long 0x88090400 Berkeley DB (Log,
>16 long 0x01000000 version 1,
>16 long 0x02000000 version 2,
>16 long 0x03000000 version 3,
>16 long 0x04000000 version 4,
>16 long 0x05000000 version 5,
>16 long 0x06000000 version 6,
>16 long 0x07000000 version 7,
>16 long 0x08000000 version 8,
>16 long 0x09000000 version 9,
>16 long 0x0a000000 version 10,
>16 long 0x0b000000 version 11,
>16 long 0x0c000000 version 12,
>16 long 0x0d000000 version 13,
>16 long 0x0e000000 version 14,
>16 long 0x0f000000 version 15,
>12 long 0x88090400 big-endian)
```

Building with multiple versions of Berkeley DB

In some cases it may be necessary to build applications which include multiple versions of Berkeley DB. Examples include applications which include software from other vendors, or applications running on a system where the system C library itself uses Berkeley DB. In such cases, the two versions of Berkeley DB may be incompatible, that is, they may have different external and internal interfaces, and may even have different underlying database formats.

To create a Berkeley DB library whose symbols won't collide with other Berkeley DB libraries (or other application or library modules, for that matter), configure Berkeley DB using the --with-uniquename=NAME configuration option, and then build Berkeley DB as usual. (Note that --with-uniquename=NAME only affects the Berkeley DB C language library build; loading multiple versions of the C++ or Java APIs will require additional work.) The modified symbol names are hidden from the application in the Berkeley DB header files, that is, there is no need for the application to be aware that it is using a special library build as long as it includes the appropriate Berkeley DB header file.

If "NAME" is not specified when configuring with --with-uniquename=NAME, a default value built from the major and minor numbers of the Berkeley DB release will be used. It is rarely necessary to specify NAME; using the major and minor release numbers will ensure that only one copy of the library will be loaded into the application unless two distinct versions really are necessary.

When distributing any library software that uses Berkeley DB, or any software which will be recompiled by users for their systems, we recommend two things: First, include the Berkeley DB release as part of your release. This will insulate your software from potential Berkeley DB API changes as well as simplifying your coding because you will only have to code to a single version of the Berkeley DB API instead of adapting at compile time to whatever version of Berkeley DB happens to be installed on the target system. Second, use --with-uniquename=NAME when configuring Berkeley DB, because that will insure that you do not unexpectedly collide with other application code or a library already installed on the target system.

Chapter 3. Debugging Applications

Introduction to debugging

Because Berkeley DB is an embedded library, debugging applications that use Berkeley DB is both harder and easier than debugging a separate server. Debugging can be harder because when a problem arises, it is not always readily apparent whether the problem is in the application, is in the database library, or is a result of an unexpected interaction between the two. Debugging can be easier because it is easier to track down a problem when you can review a stack trace rather than deciphering interprocess communication messages. This chapter is intended to assist you with debugging applications and reporting bugs to us so that we can provide you with the correct answer or fix as quickly as possible.

When you encounter a problem, there are a few general actions you can take:

Review the Berkeley DB error output:

If an error output mechanism has been configured in the Berkeley DB environment, additional run-time error messages are made available to the applications. If you are not using an environment, it is well worth modifying your application to create one so that you can get more detailed error messages. See Run-time error information (page 11) for more information on configuring Berkeley DB to output these error messages.

Review the options available for the DB_ENV->set_verbose() method:

Look to see if it offers any additional informational and/or debugging messages that might help you understand the problem.

Add run-time diagnostics:

You can configure and build Berkeley DB to perform run-time diagnostics. (By default, these checks are not done because they can seriously impact performance.) See Compile-time configuration (page 10) for more information.

Apply all available patches:

Before reporting a problem in Berkeley DB, please upgrade to the latest Berkeley DB release, if possible, or at least make sure you have applied any updates available for your release from the Berkeley DB web site .

Run the test suite:

If you see repeated failures or failures of simple test cases, run the Berkeley DB test suite to determine whether the distribution of Berkeley DB you are using was built and configured correctly.

Compile-time configuration

There are three compile-time configuration options that assist in debugging Berkeley DB and Berkeley DB applications:

--enable-debug

If you want to build Berkeley DB with **-g** as the C and C++ compiler flag, enter --enable-debug as an argument to configure. This will create Berkeley DB with debugging symbols, as well as load various Berkeley DB routines that can be called directly from a debugger to display database page content, cursor queues, and so forth. (Note that the **-O** optimization flag will still be specified. To compile with only the **-g**, explicitly set the `CFLAGS` environment variable before configuring.)

--enable-diagnostic

If you want to build Berkeley DB with debugging run-time sanity checks and with DIAGNOSTIC #defined during compilation, enter --enable-diagnostic as an argument to configure. This will cause a number of special checks to be performed when Berkeley DB is running. This flag should not be defined when configuring to build production binaries because it degrades performance.

--enable-umrw

When compiling Berkeley DB for use in run-time memory consistency checkers (in particular, programs that look for reads and writes of uninitialized memory), use --enable-umrw as an argument to configure. This guarantees, among other things, that Berkeley DB will completely initialize allocated pages rather than initializing only the minimum necessary amount.

Run-time error information

Normally, when an error occurs in the Berkeley DB library, an integer value (either a Berkeley DB specific value or a system `errno` value) is returned by Berkeley DB. In some cases, however, this value may be insufficient to completely describe the cause of the error, especially during initial application debugging.

Most Berkeley DB errors will result in additional information being written to a standard file descriptor or output stream. Additionally, Berkeley DB can be configured to pass these verbose error messages to an application function. There are four methods intended to provide applications with additional error information: DB_ENV->set_errcall(), DB_ENV->set_errfile(), DB_ENV->set_errpfx() and DB_ENV->set_verbose().

The Berkeley DB error-reporting facilities do not slow performance or significantly increase application size, and may be run during normal operation as well as during debugging. Where possible, we recommend these options always be configured and the output saved in the filesystem. We have found that this often saves time when debugging installation or other system-integration problems.

In addition, there are three methods to assist applications in displaying their own error messages: db_strerror(), DB_ENV->err(), and `DB_ENV->errx()`. The first is a superset of the ANSI C strerror function, and returns a descriptive string for any error return from the Berkeley DB library. The DB_ENV->err() and `DB_ENV->errx()` methods use the error message configuration options described previously to format and display error messages to appropriate output devices.

Reviewing Berkeley DB log files

If you are running with transactions and logging, the db_printlog utility can be a useful debugging aid. The db_printlog utility will display the contents of your log files in a human readable (and machine-readable) format.

The db_printlog utility will attempt to display any and all log files present in a designated db_home directory. For each log record, the db_printlog utility will display a line of the form:

```
[22][28]db_big: rec: 43 txnid 80000963 prevlsn [21][10483281]
```

The opening numbers in square brackets are the *log sequence number* (*LSN*) of the log record being displayed. The first number indicates the log file in which the record appears, and the second number indicates the offset in that file of the record.

The first character string identifies the particular log operation being reported. The log records corresponding to particular operations are described following. The rest of the line consists of name/value pairs.

The rec field indicates the record type (this is used to dispatch records in the log to appropriate recovery functions).

The txnid field identifies the transaction for which this record was written. A txnid of 0 means that the record was written outside the context of any transaction. You will see these most frequently for checkpoints.

Finally, the prevlsn contains the LSN of the last record for this transaction. By following prevlsn fields, you can accumulate all the updates for a particular transaction. During normal abort processing, this field is used to quickly access all the records for a particular transaction.

After the initial line identifying the record type, each field of the log record is displayed, one item per line. There are several fields that appear in many different records and a few fields that appear only in some records.

The following table presents each currently written log record type with a brief description of the operation it describes. Any of these record types may have the string "_debug" appended if they were written because DB_TXN_NOT_DURABLE was specified and the system was configured with --enable-diagnostic.

Log Record Type	Description
bam_adj	Used when we insert/remove an index into/ from the page header of a Btree page.
bam_cadjust	Keeps track of record counts in a Btree or Recno database.
bam_cdel	Used to mark a record on a page as deleted.
bam_curadj	Used to adjust a cursor location when a nearby record changes in a Btree database.
bam_merge	Used to merge two Btree database pages during compaction.
bam_pgno	Used to replace a page number in a Btree record.
bam_rcuradj	Used to adjust a cursor location when a nearby record changes in a Recno database.
bam_relink	Fix leaf page prev/next chain when a page is removed.
bam_repl	Describes a replace operation on a record.
bam_root	Describes an assignment of a root page.

Log Record Type	Description
bam_rsplit	Describes a reverse page split.
bam_split	Describes a page split.
crdel_inmem_create	Record the creation of an in-memory named database.
crdel_inmem_remove	Record the removal of an in-memory named database.
crdel_inmem_rename	Record the rename of an in-memory named database.
crdel_metasub	Describes the creation of a metadata page for a subdatabase.
db_addrem	Add or remove an item from a page of duplicates.
db_big	Add an item to an overflow page (*overflow pages* contain items too large to place on the main page)
db_cksum	Unable to checksum a page.
db_debug	Log debugging message.
db_noop	This marks an operation that did nothing but update the LSN on a page.
db_ovref	Increment or decrement the reference count for a big item.
db_pg_alloc	Indicates we allocated a page to a database.
db_pg_free	Indicates we freed a page (freed pages are added to a freelist and reused).
db_pg_freedata	Indicates we freed a page that still contained data entries (freed pages are added to a freelist and reused.)
db_pg_init	Indicates we reinitialized a page during a truncate.
db_pg_sort	Sort the free page list and free pages at the end of the file.
dbreg_register	Records an open of a file (mapping the filename to a log-id that is used in subsequent log operations).
fop_create	Create a file in the file system.
fop_file_remove	Remove a name in the file system.
fop_remove	Remove a file in the file system.
fop_rename	Rename a file in the file system.
fop_write	Write bytes to an object in the file system.

Log Record Type	Description
ham_chgpg	Used to adjust a cursor location when a Hash page is removed, and its elements are moved to a different Hash page.
ham_copypage	Used when we empty a bucket page, but there are overflow pages for the bucket; one needs to be copied back into the actual bucket.
ham_curadj	Used to adjust a cursor location when a nearby record changes in a Hash database.
ham_groupalloc	Allocate some number of contiguous pages to the Hash database.
ham_insdel	Insert/delete an item on a Hash page.
ham_metagroup	Update the metadata page to reflect the allocation of a sequence of contiguous pages.
ham_newpage	Adds or removes overflow pages from a Hash bucket.
ham_replace	Handle updates to records that are on the main page.
ham_splitdata	Record the page data for a split.
qam_add	Describes the actual addition of a new record to a Queue.
qam_del	Delete a record in a Queue.
qam_delext	Delete a record in a Queue with extents.
qam_incfirst	Increments the record number that refers to the first record in the database.
qam_mvptr	Indicates we changed the reference to either or both of the first and current records in the file.
txn_child	Commit a child transaction.
txn_ckp	Transaction checkpoint.
txn_recycle	Transaction IDs wrapped.
txn_regop	Logs a regular (non-child) transaction commit.
txn_xa_regop	Logs a prepare message.

Augmenting the Log for Debugging

When debugging applications, it is sometimes useful to log not only the actual operations that modify pages, but also the underlying Berkeley DB functions being executed. This form of logging can add significant bulk to your log, but can permit debugging application errors that are almost impossible to find any other way. To turn on these log messages, specify the --

enable-debug_rop and --enable-debug_wop configuration options when configuring Berkeley DB. See Configuring Berkeley DB (page 38) for more information.

Extracting Committed Transactions and Transaction Status

Sometimes, it is helpful to use the human-readable log output to determine which transactions committed and aborted. The awk script, commit.awk, (found in the db_printlog directory of the Berkeley DB distribution) allows you to do just that. The following command, where log_output is the output of db_printlog, will display a list of the transaction IDs of all committed transactions found in the log:

```
awk -f commit.awk log_output
```

If you need a complete list of both committed and aborted transactions, then the script status.awk will produce it. The syntax is as follows:

```
awk -f status.awk log_output
```

Extracting Transaction Histories

Another useful debugging aid is to print out the complete history of a transaction. The awk script txn.awk allows you to do that. The following command line, where log_output is the output of the db_printlog utility and txnlist is a comma-separated list of transaction IDs, will display all log records associated with the designated transaction ids:

```
awk -f txn.awk TXN=txnlist log_output
```

Extracting File Histories

The awk script fileid.awk allows you to extract all log records that refer to a designated file. The syntax for the fileid.awk script is the following, where log_output is the output of db_printlog and fids is a comma-separated list of fileids:

```
awk -f fileid.awk PGNO=fids log_output
```

Extracting Page Histories

The awk script pgno.awk allows you to extract all log records that refer to designated page numbers. However, because this script will extract records with the designated page numbers for all files, it is most useful in conjunction with the fileid script. The syntax for the pgno.awk script is the following, where log_output is the output of db_printlog and pgnolist is a comma-separated list of page numbers:

```
awk -f pgno.awk PGNO=pgnolist log_output
```

Other log processing tools

The awk script count.awk prints out the number of log records encountered that belonged to some transaction (that is, the number of log records excluding those for checkpoints and non-transaction-protected operations).

The script range.awk will extract a subset of a log. This is useful when the output of db_printlog utility is too large to be reasonably manipulated with an editor or other tool. The

syntax for range.awk is the following, where **sf** and **so** represent the LSN of the beginning of the sublog you want to extract, and **ef** and **eo** represent the LSN of the end of the sublog you want to extract:

```
awk -f range.awk START_FILE=sf START_OFFSET=so END_FILE=ef \
END_OFFSET=eo log_output
```

Chapter 4. Building Berkeley DB for Windows

This chapter contains general instructions on building Berkeley DB for specific windows platforms using specific compilers. The Windows FAQ (page 30) also contains helpful information.

The `build_windows` directory in the Berkeley DB distribution contains project files for Microsoft Visual Studio:

Project File	Description
Berkeley_DB.sln	Visual Studio 2005 (8.0) workspace
*.vcproj	Visual Studio 2005 (8.0) projects
Berkeley_DB_vs2010.sln	Visual Studio 2010 workspace
*.vcxproj	Visual Studio 2010 projects

These project files can be used to build Berkeley DB for the following platforms: Windows NT/2K/XP/2003/Vista and Windows7; and 64-bit Windows XP/2003/Vista and Windows7.

Building Berkeley DB for 32 bit Windows

Visual C++ .NET 2010

1. Choose *File* -> *Open* -> *Project/Solution*.... In the `build_windows` directory, select `Berkeley_DB_vs2010.sln` and click Open.
2. Choose the desired project configuration from the drop-down menu on the tool bar (either Debug or Release).
3. Choose the desired platform configuration from the drop-down menu on the tool bar (usually Win32 or x64).
4. To build, right-click on the `Berkeley_DB_vs2010` solution and select Build Solution.

Visual C++ .NET 2008

1. Choose *File* -> *Open* -> *Project/Solution*.... In the `build_windows` directory, select `Berkeley_DB.sln` and click Open.
2. The *Visual Studio Conversion Wizard* will open automatically. Click the *Finish* button.
3. On the next screen click the *Close* button.
4. Choose the desired project configuration from the drop-down menu on the tool bar (either Debug or Release).
5. Choose the desired platform configuration from the drop-down menu on the tool bar (usually Win32 or x64).

6. To build, right-click on the Berkeley_DB solution and select Build Solution.

Visual C++ .NET 2005

1. Choose *File -> Open -> Project/Solution...*. In the `build_windows` directory, select `Berkeley_DB.sln` and click Open
2. Choose the desired project configuration from the drop-down menu on the tool bar (either Debug or Release).
3. Choose the desired platform configuration from the drop-down menu on the tool bar (usually Win32 or x64).
4. To build, right-click on the Berkeley_DB solution and select Build Solution.

Build results

The results of your build will be placed in one of the following Berkeley DB subdirectories, depending on the configuration that you chose:

```
build_windows\Win32\Debug
build_windows\Win32\Release
build_windows\Win32\Debug_static
build_windows\Win32\Release_static
```

When building your application during development, you should normally use compile options "Debug Multithreaded DLL" and link against `build_windows\Debug\libdb51d.lib`. You can also build using a release version of the Berkeley DB libraries and tools, which will be placed in `build_windows\Win32\Release\libdb51.lib`. When linking against the release build, you should compile your code with the "Release Multithreaded DLL" compile option. You will also need to add the `build_windows` directory to the list of include directories of your application's project, or copy the Berkeley DB include files to another location.

Building Berkeley DB for 64-bit Windows

The following procedure can be used to build natively on a 64-bit system or to cross-compile from a 32-bit system.

When building 64-bit binaries, the output directory will be one of the following Berkeley DB subdirectories, depending upon the configuration that you chose:

```
build_windows\x64\Debug
build_windows\x64\Release
build_windows\x64\Debug_static
build_windows\x64\Release_static
```

x64 build with Visual Studio 2005 or newer

1. Follow the build instructions for your version of Visual Studio, as described in Building Berkeley DB for 32 bit Windows (page 17).

2. Select *x64* from the *Platform Configuration* dropdown.

3. Right click on *Solution 'Berkeley_DB'* in the Solution Explorer, and select *Build Solution*

Building Berkeley DB with Cygwin

To build Berkeley DB with Cygwin, follow the instructions in Building for UNIX/POSIX (page 37).

Building the C++ API

C++ support is built automatically on Windows.

Building the C++ STL API

In the project list of the `Berkeley_DB.sln` solution, build the "db_stl" project and "db_stl_static" project to build STL API as a dynamic or static library respectively. And in your application, you should link this library file as well as the Berkeley DB library file to your application. The STL API library file is by default always put at the same location as the Berkeley DB library file.

And you need to include the STL API header files in your application code. If you are using the Berkeley DB source tree, the header files are in <Berkeley DB Source Root >/stl directory; If you are using the pre-built installed version, these header files are in < Berkeley DB Installed Directory>/include, as well as the db.h and db_cxx.h header files.

Building the Java API

Java support is not built automatically. The following instructions assume that you have installed the Sun Java Development Kit in `d:\java`. Of course, if you installed elsewhere or have different Java software, you will need to adjust the pathnames accordingly.

1. Set your include directories.

   - In Visual Studio 2005/Visual Studio 2008 - Choose *Tools -> Options -> Projects -> VC++ Directories*. Under the "Show directories for" pull-down, select "Include files". Add the full pathnames for the `d:\java\include` and `d:\java\include\win32` directories. Then click OK.

   - In Visual Studio 2010 - Right-click db_java project, choose *Properties->Configuration Properties-> VC++ Directories->Include Directories*. Add the full pathnames for the `d:\java\include` and `d:\java\include\win32` directories. Then click OK.

   These are the directories needed when including jni.h.

2. Set the executable files directories.

   - In Visual Studio 2005/Visual Studio 2008 - Choose *Tools -> Options -> Projects -> VC++ Directories*. Under the "Show directories for" pull-down, select "Executable files". Add the full pathname for the `d:\java\bin` directory, then click OK.

- In Visual Studio 2010 - Right-click db_java project, choose *Properties->Configuration Properties-> VC++ Directories->Executable Directories*. Add the full pathnames for the `d:\java\bin` directories. Then click OK.

This is the directory needed to find javac.

3. Set the build type to Release or Debug in the drop-down on the tool bar.

4. To build, right-click on db_java and select Build. This builds the Java support library for Berkeley DB and compiles all the java files, placing the resulting `db.jar` and `dbexamples.jar` files in one of the following Berkeley DB subdirectories, depending on the configuration that you chose:

   ```
   build_windows\Win32\Debug
   build_windows\Win32\Release
   ```

Building the C# API

The C# support is built by a separate Visual Studio solution and requires version 2.0 (or higher) of the .NET platform. In Visual Studio 2005/Visual Studio 2008, the solution is `build_windows\BDB_dotnet.sln`; in Visual Studio 2010, the solution is `build_windows\BDB_dotnet_vs2010.sln`.

By default, the solution will build the native libraries, the managed assembly and all example programs. The NUnit tests need to be built explicitly because of their dependence upon the NUnit assembly. The native libraries will be placed in one of the following subdirectories, depending upon the chosen configuration:

```
build_windows\Win32\Debug
build_windows\Win32\Release
build_windows\x64\Debug
build_windows\x64\Release
```

The managed assembly and all C# example programs will be placed in one of the following subdirectories, depending upon the chosen configuration:

```
build_windows\AnyCPU\Debug
build_windows\AnyCPU\Release
```

The native libraries need to be locatable by the .NET platform, meaning they must be copied into an application's directory, the Windows or System directory, or their location must be added to the PATH environment variable. The example programs demonstrate how to programmatically edit the PATH variable.

Building the SQL API

SQL support is built as part of the default build on Windows. For information on the build instructions, see Building Berkeley DB for Windows (page 17).

The SQL library is built as `libdb_sql51.dll` in the Release mode or `libdb_sql51d.dll` in the Debug mode. An SQL command line interpreter called `dbsql.exe` is also built.

Binary Compatibility With SQLite

`libdb_sql51.dll` is compatible with `sqlite3.dll`. You can copy `libdb_sql51.dll` to `sqlite3.dll` and `dbsql.exe` to `sqlite3.exe`, and use these applications as a replacement for the standard SQLite binaries with the same names. However, if you want to do this, then any legacy data in use by those tools must be migrated from the standard SQLite database to a Berkeley DB SQL database *before* you replace the standard SQLite dll and executable with the Berkeley DB equivalent. For information on migrating data from standard SQLite databases to a Berkeley DB SQL database, see the *Berkeley DB Getting Started with the SQL APIs* guide.

Warning

Rename your dlls and executables to the standard SQLite names with *extreme* care. Doing this will cause all existing tools to break that currently have data stored in a standard SQLite database.

For best results, rename your dlls and command line tool to use the standard SQLite names only if you know there are no other tools on your production platform that rely on standard SQLite.

Setting Preprocessor Flags

By default, Berkeley DB SQL generates each table as a subdatabase in a single file. To generate each table in a separate file, specify *BDBSQL_FILE_PER_TABLE* in *Preprocessor Definitions* of the `db_sql` project.

When this option is enabled, the SQL database name is used as a directory name. This directory contains one file for the metadata and one file each for every table created by the SQL API. Do not add or delete files from the database directory. Adding or deleting files may corrupt the database. To backup just the metadata (schema), make a copy of the `metadata` and `table00001` files from the database directory. Make a new copy whenever the schema is changed.

Enabling Extensions

The Berkeley DB SQL API provides extensions such as full text search and R-Tree index. To enable these extensions, do the following:

1. Open the Berkeley DB solution in Visual Studio.
2. Specify *SQLITE_ENABLE_FTS3* or *SQLITE_ENABLE_RTREE* in *Preprocessor Definitions* of the `db_sql` project.
3. Re-build the `db_sql` project.

See the SQLite Documentation for more information on full text search and R-Tree.

Building the JDBC Driver

This section describes the steps to build the JDBC driver.

1. Configure your build environment. For information on how to configure to build Java applications, see Building the Java API (page 19).
2. Build the SQL project in Debug mode.
3. Open Visual Studio.
4. Select *File* -> *Add* -> *Existing Project*.
5. Select `build_windows/db_sql_jdbc.vcproj` and add it to the Berkeley_DB solution. This adds the `db_sql_jdbc` Visual Studio project to the Berkeley_DB solution file.
6. Build the `db_sql_jdbc` project in Visual Studio.

You can test the build by entering the following commands from the `db\build_windows\Win32\Debug` directory:

```
javac -cp ".;jdbc.jar" -d . ..\..\..\sql\jdbc\test3.java
java -cp ".;jdbc.jar" test3
```

When building the JDBC driver, if you may see an error message:
"SQLite.JDBC2x.JDBCConnection is not abstract and does not override abstract method in java.sql.Connection".

This means that your Java environment requires `JDBC2z.*` instead of `JDBC2x.*`. To resolve this problem, do the following:

- In the Solution Explorer, right-click the `db_sql_jdbc` project and select *properties*.
- In the *Configuration Properties* -> *Build Events* -> *Pre-Build Event* section, alter the command to refer to `JDBC2z` instead of `JDBC2x`.

Using the JDBC Driver

This section describes the steps to download, build, and run sample programs using the built JDBC driver.

Downloading JDBC Sample Code

The download link for JDBC sample code is available on the Oracle Sun Developer Network (SDN). You can identify the link by the "JDBC programming examples from all three editions (ZIP format)" text beside it.

Modifying Sample Code

Before running the sample code, do the following:

1. Unzip the file containing the sample code to a new directory (for example, jdbc_ex).
2. Substitute `jdbc:sqlite:/<db-file-name>` for the generic JDBC URL that appears in the code. That is, put `jdbc:sqlite:/<db-file-name>` between the quotation marks in the line:

```
String url = "jdbc:mySubprotocol:myDataSource";
```

Note: The <db-file-name> can either be an absolute path name like `"D:\\jdbc_ex_db\\myDataSource"`, or a relative path-file-name like `"..\\jdbc_ex_db\myDataSource"`, or a file name, like `"myDataSource"`, in which the database file will be stored at the current directory.

3. Substitute `SQLite.JDBCDriver` for `myDriver.ClassName` in the line: `Class.forName("myDriver.ClassName");`

4. Substitute the username and password you use for your database in the following: `"myLogin", "myPassword"`.

   This is optional.

5. If your JDK version is above 1.5, change the variable name `enum` in `OutputApplet.java` to some other variable name because, as of JDK release 5 `enum` is a keyword and can not be used as an identifier.

Building and Running the JDBC Sample code

See Building the JDBC Driver (page 21) for instructions about building JDBC driver.

To build and run the JDBC examples do the following:

1. In the `db\build_windows\Win32\Debug` directory, run following commands:

```
$ javac -classpath ".;jdbc.jar" -d . \path\to\jdbc_ex\*.java
$ java  -classpath ".;jdbc.jar" <ClassName, eg. CreateCoffees>
```

2. After you run the CreateCoffees example, use the `dbsql` executable to open the `myDataSource` database file and check if the table `COFFEES` has been successfully created in the database.

```
$ dbsql myDataSourcedbsql> .tables
COFFEES
dbsql> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE COFFEES (COF_NAME varchar(32),/
SUP_ID int, PRICE float, SALES int, TOTAL int);
COMMIT;
dbsql>
```

3. Repeat step 2 to run other examples.
   Note: Some examples like AutoGenKeys are not yet supported by BDB JDBC driver. The `SQLFeatureNotSupportedException` is displayed for those unsupported examples.

Building the ODBC Driver

This section describes the steps required to build the ODBC driver.

Configuring Your System

To configure your system prior to building the ODBC driver, do the following:

1. Download and install the latest SQLite ODBC driver Windows installer package for 32 bit Windows or 64 bit Windows.
2. Download and install the latest Microsoft Data Access Components (MDAC) SDK. The MDAC SDK is only required for testing the installation.

Building the Library

1. Build the SQL project in Release mode. See Building the SQL API (page 20).
2. Open Visual Studio.
3. Load the Berkeley_DB solution file into Visual Studio.
4. Set the build target to *Release*
5. Build the solution.
6. Select *File* -> *Add* -> *Existing Project*.
7. Select `build_windows/db_sql_odbc.vcproj` and add it to the Berkeley_DB solution. This adds the `db_sql_odbc` Visual Studio project to the Berkeley_DB solution file.
8. Build the `db_sql_odbc` project. This can be done by right-clicking the `db_sql_odbc` project in the project explorer panel, and selecting `build`.

The `sqlite3odbc.dll`, `libdb_sql51.dll` and `libdb51.dll` files are now built.

Installing the Library

Copy the dll files built in the *Building the Library* section to the Windows system folder.

The Windows system folder is different on different systems, but is often `C:\WINDOWS\System32`.

Testing the ODBC Install

The steps to verify that the installed driver works are as follows:

1. Open the Unicode ODBCTest application. On Windows XP: *Windows start* -> *Microsoft Data Access SDK 2.8* -> *ODBCTest (Unicode, x86)*.
2. Select the *Conn* -> *Full Connect*... menu item.
3. Select `SQLite3 Datasource` and click `OK`.

4. Select the *Stmt* -> *SQLExecDirect*... menu item.
5. Enter `CREATE TABLE t1(x);` in the `Statement` text box and click `OK`.
6. Verify that no error messages were output to the error window.

Using the ADO.NET Driver

This section describes the steps to use the ADO.NET driver made by phxsoftware. The Berkeley DB development team has tested version 1.0.66.0 of the ADO.NET driver.

1. Download and unzip `SQLite-1.0.66.0-binaries.zip`. The .NET binary, `System.Data.SQLite.dll`, is located in the `ManagedOnly` directory.
2. Build the Berkeley DB SQL API. For more information, see Building the SQL API (page 20). If you have installed the Windows binaries, you can ignore this step.
3. Copy `System.Data.SQLite.dll and libdb51.dll` into a directory. If you have installed the Windows binaries, copy `System.Data.SQLite.dll` into the `bin` directory of the installation.
4. Copy `libdb_sql51.dll` to `sqlite3.dll`. Place `sqlite3.dll` in the same directory as `System.Data.SQLite.dll`.
5. Add this directory to the environment variable PATH. Alternatively, you can copy the dlls into the directory from which the applications is run. If you have installed binaries, the environment variable PATH is already set to include the installation's `bin` directory.

Running the ADO.NET Test Suite

This section describes how to run the test suite made by phxsoftware.

1. Download the driver and build Berkeley DB as described above.
2. Unzip and copy `libdb_sql51.dll` and `libdb51.dll` into the `ManagedOnly` directory.
3. Copy `libdb_sql51.dll` to `sqlite3.dll`.
4. Run `test.exe` from the `ManagedOnly` directory.

When running the test suite, exclude `LockTest` and `MultithreadingTest` due to differences between SQLite locking and Berkeley DB locking. For more information, see the Berkeley DB SQL Guide. You should also exclude `FunctionWithCollation` and `FunctionWithCollation2` tests because they use extension functions which are not a part of the default SQLite distribution. For more information, see http://www.sqlite.org/contrib.

Building the Tcl API

Tcl support is not built automatically. See Loading Berkeley DB with Tcl for information on sites from which you can download Tcl and which Tcl versions are compatible with Berkeley DB. These notes assume that Tcl is installed as `d:\tcl`, but you can change that if you want.

The Tcl library must be built as the same build type as the Berkeley DB library (both Release or both Debug). We found that the binary release of Tcl can be used with the Release configuration of Berkeley DB, but you will need to build Tcl from sources for the Debug configuration. Before building Tcl, you will need to modify its makefile to make sure that you are building a debug version, including thread support. This is because the set of DLLs linked into the Tcl executable must match the corresponding set of DLLs used by Berkeley DB.

1. Set the include directories.

   - In Visual Studio 2005/Visual Studio 2008 - Choose *Tools -> Options -> Projects -> VC++ Directories*. Under the "Show directories for" pull-down, select "Include files". Add the full pathname for `d:\tcl\include`, then click OK.

   - In Visual Studio 2010 - Right-click db_java project, choose *Properties->Configuration Properties-> VC++ Directories->Include Directories*. Add the full pathnames for `d:\tcl\include`, then click OK.

   This is the directory that contains `tcl.h`.

2. Set the library files directory.

   - In Visual Studio 2005/Visual Studio 2008 - Choose *Tools -> Options -> Projects -> VC++ Directories*. Under the "Show directories for" pull-down, select "Library files". Add the full pathname for the `d:\tcl\lib` directory, then click OK.

   - In Visual Studio 2010 - Right-click db_java project, choose *Properties->Configuration Properties-> VC++ Directories->Library Directories*. Add the full pathname for the `d:\tcl\lib` directory, then click OK.

   This is the directory needed to find `tcl85g.lib` (or whatever the library is named in your distribution).

3. Set the build type to Release or Debug in the drop-down on the tool bar.

4. To build, right-click on db_tcl and select Build. This builds the Tcl support library for Berkeley DB, placing the result into one of the following Berkeley DB subdirectories, depending upon the configuration that you chose:

   ```
   build_windows\Win32\Debug\libdb_tcl51d.dll
   build_windows\Win32\Release\libdb_tcl51.dll
   ```

If you use a version different from Tcl 8.5.x you will need to change the name of the Tcl library used in the build (for example, `tcl85g.lib`) to the appropriate name. To do this, right click on *db_tcl*, go to *Properties -> Linker -> Input -> Additional dependencies* and change `tcl85g.lib` to match the Tcl version you are using.

Distributing DLLs

When distributing applications linked against the DLL (not static) version of the library, the DLL files you need will be found in one of the following Berkeley DB subdirectories, depending upon the configuration that you chose:

`build_windows\Win32\Debug`

```
build_windows\Win32\Release
build_windows\Win32\Debug_static
build_windows\Win32\Release_static
build_windows\x64\Debug
build_windows\x64\Release
build_windows\x64\Debug_static
build_windows\x64\Release_static
```

You may also need to redistribute DLL files needed for the compiler's runtime. Generally, these runtime DLL files can be installed in the same directory that will contain your installed Berkeley DB DLLs. This directory may need to be added to your System PATH environment variable. Check your compiler's license and documentation for specifics on redistributing runtime DLLs.

Building a small memory footprint library

For applications that don't require all of the functionality of the full Berkeley DB library, an option is provided to build a static library with certain functionality disabled. In particular, cryptography, hash and queue access methods, replication and verification are all turned off. This can reduce the memory footprint of Berkeley DB significantly.

In general on Windows systems, you will want to evaluate the size of the final application, not the library build. The Microsoft LIB file format (like UNIX archives) includes copies of all of the object files and additional information. The linker rearranges symbols and strips out the overhead, and the resulting application is much smaller than the library. There is also a Visual C++ optimization to "Minimize size" that will reduce the library size by a few percent.

A Visual C++ project file called `db_small` is provided for this small memory configuration. During a build, static libraries are created in `Release` or `Debug`, respectively. The library name is `libdb_small48sd.lib` for the debug build, or `libdb_small48s.lib` for the release build.

For assistance in further reducing the size of the Berkeley DB library, or in building small memory footprint libraries on other systems, please contact Berkeley DB support.

Running the test suite under Windows

To build the test suite on Windows platforms, you will need to configure Tcl support. You will also need sufficient main memory (at least 64MB), and disk (around 250MB of disk will be sufficient).

Building the software needed by the tests

The test suite must be run against a Debug version of Berkeley DB, so you will need a Debug version of the Tcl libraries. This involves building Tcl from its source. See the Tcl sources for more information. Then build the Tcl API - see Building the Tcl API (page 25) for details.

Visual Studio 2005 or newer

To build for testing, perform the following steps:

1. Open the Berkeley DB solution.
2. Ensure that the target configuration is Debug
3. Right click the *db_tcl* project in the Solution Explorer, and select *Build*.
4. Right click the *db_test* project in the Solution Explorer, and select *Build*.

Running the test suite under Windows

Before running the tests for the first time, you must edit the file `include.tcl` in your build directory and change the line that reads:

```
set tclsh_path SET_YOUR_TCLSH_PATH
```

You will want to use the location of the `tclsh` program (be sure to include the name of the executable). For example, if Tcl is installed in `d:\tcl`, this line should be the following:

```
set tclsh_path d:\tcl\bin\tclsh85g.exe
```

If your path includes spaces be sure to enclose it in quotes:

```
set tclsh_path "c:\Program Files\tcl\bin\tclsh85g.exe"
```

Make sure that the path to Berkeley DB's tcl library is in your current path. On Windows NT/2000/XP, edit your PATH using the My Computer -> Properties -> Advanced -> Environment Variables dialog. On earlier versions of Windows, you may find it convenient to add a line to c:\AUTOEXEC.BAT:

```
SET PATH=%PATH%;c:\db\build_windows
```

Then, in a shell of your choice enter the following commands:

1. cd build_windows
2. run `d:\tcl\bin\tclsh85g.exe`, or the equivalent name of the Tcl shell for your system.

   You should get a "%" prompt.
3. % source ../test/tcl/test.tcl
   If no errors occur, you should get a "%" prompt.

You are now ready to run tests in the test suite; see Running the test suite for more information.

Building the software needed by the SQL tests

The SQL test suite must be run against a Debug version of Berkeley DB, so you need a Debug version of the Tcl libraries. This involves building Tcl from its source. See the Tcl sources for more information. Then build the Tcl API - see Building the Tcl API (page 25) for details.

Before building for SQL tests, build the db_tcl and db_sql_testfixture projects. This requires Tcl 8.4 or above. If you are using a later version of Tcl, edit the Tcl library that db_tcl and db_sql_testfixture link to.

To do this right click the *db_tcl/db_sql_testfixture* project, select *Properties->Configuration Properties->Linker->Input->Additional Dependencies* and edit the Tcl library, *tcl85g.lib*, to match the version you are using.

Building the db_sql_testfixture project builds the testfixture.exe program in `../build_windows/Win32/Debug`. It also builds the projects db and db_sql, on which it depends.

Visual Studio 2005 or newer

To build for testing, perform the following steps:

1. Open the Berkeley DB solution.
2. Ensure that the target configuration is Debug.
3. Right click the *db_tcl* project in the Solution Explorer, and select *Build*.
4. Right click the *db_sql_testfixture* project in the Solution Explorer, and select *Build*.

To test extensions, specify the following in the *Preprocessor Definitions* of the *db_sql_testfixture* project:

- `SQLITE_ENABLE_FTS3` to enable the full text search layer
- `SQLITE_ENABLE_RTREE` to enable the R-Tree layer

Windows notes

If a system memory environment is closed by all processes, subsequent attempts to open it will return an error. To successfully open a transactional environment in this state, recovery must be run by the next process to open the environment. For non-transactional environments, applications should remove the existing environment and then create a new database environment.

1. Berkeley DB does not support the Windows/95, Windows/98 or Windows/ME platforms.
2. On Windows, system paging file memory is freed on last close. For this reason, multiple processes sharing a database environment created using the DB_SYSTEM_MEM flag must arrange for at least one process to always have the environment open, or alternatively that any process joining the environment be prepared to re-create it.
3. When using the DB_SYSTEM_MEM flag, Berkeley DB shared regions are created without ACLs, which means that the regions are only accessible to a single user. If wider sharing is appropriate (for example, both user applications and Windows/NT service applications need to access the Berkeley DB regions), the Berkeley DB code will need to be modified to create the shared regions with the correct ACLs. Alternatively, by not specifying the DB_SYSTEM_MEM flag, filesystem-backed regions will be created instead, and the permissions on those files may be directly specified through the DB_ENV->open() method.
4. Applications that operate on wide character strings can use the Windows function WideCharToMultiByte with the code page CP_UTF8 to convert paths to the form expected

by Berkeley DB. Internally, Berkeley DB calls MultiByteToWideChar on paths before calling Windows functions.

5. Various Berkeley DB methods take a **mode** argument, which is intended to specify the underlying file permissions for created files. Berkeley DB currently ignores this argument on Windows systems.

   It would be possible to construct a set of security attributes to pass to **CreateFile** that accurately represents the mode. In the worst case, this would involve looking up user and all group names, and creating an entry for each. Alternatively, we could call the **_chmod** (partial emulation) function after file creation, although this leaves us with an obvious race.

   Practically speaking, however, these efforts would be largely meaningless on a FAT file system, which only has a "readable" and "writable" flag, applying to all users.

Windows FAQ

1. **My Win* C/C++ application crashes in the Berkeley DB library when Berkeley DB calls fprintf (or some other standard C library function).**

   You should be using the "Debug Multithreaded DLL" compiler option in your application when you link with the build_windows\Debug\libdb48d.lib library (this .lib file is actually a stub for libdb48d.DLL). To check this setting in Visual C++, choose the *Project/Settings* menu item and select *Code Generation* under the tab marked *C/C++*; and see the box marked *Use runtime library*. This should be set to *Debug Multithreaded DLL*. If your application is linked against the static library, build_windows\Debug\libdb48sd.lib; then, you will want to set *Use runtime library* to *Debug Multithreaded*.

   Setting this option incorrectly can cause multiple versions of the standard libraries to be linked into your application (one on behalf of your application, and one on behalf of the Berkeley DB library). That violates assumptions made by these libraries, and traps can result.

   Also, using different Visual Studio compilers in the application and libraries can lead to a crash. So rebuild the application with the same Visual C++ version as that of the library.

2. **Why are the build options for DB_DLL marked as "Use MFC in a Shared DLL"? Does Berkeley DB use MFC?**

   Berkeley DB does not use MFC at all. It does however, call malloc and free and other facilities provided by the Microsoft C runtime library. We found in our work that many applications and libraries are built assuming MFC, and specifying this for Berkeley DB solves various interoperation issues, and guarantees that the right runtime libraries are selected. Note that because we do not use MFC facilities, the MFC library DLL is not marked as a dependency for libdb.dll, but the appropriate Microsoft C runtime is.

3. **How can I build Berkeley DB for MinGW?**

   Follow the instructions in Building for UNIX/POSIX (page 37), and specify the --enable-mingw option to the configuration script. This configuration option currently only builds

static versions of the library, it does not yet build a DLL version of the library, and file sizes are limited to 2GB (2^32 bytes.)

4. **How can I build a Berkeley DB for Windows 98/ME?**

   Windows 98/ME is no longer supported by Berkeley DB. The following is therefore only of interest to historical users of Berkeley DB.

   By default on Windows, Berkeley DB supports internationalized filenames by treating all directory paths and filenames passed to Berkeley DB methods as UTF-8 encoded strings. All paths are internally converted to wide character strings and passed to the wide character variants of Windows system calls.

   This allows applications to create and open databases with names that cannot be represented with ASCII names while maintaining compatibility with applications that work purely with ASCII paths.

   Windows 98 and ME do not support Unicode paths directly. To build for those versions of Windows, either:

   - Follow the instructions at Microsoft's web site.
   - Open the workspace or solution file with Visual Studio. Then open the Project properties/settings section for the project you need to build (at least db_dll). In the *C/C++->Preprocessor->Preprocessor Definitions* section, remove *_UNICODE* and *UNICODE* entries. Add in an entry of *_MBCS*. Build the project as normal.

   The ASCII builds will also work on Windows NT/2K/XP/2003 and Windows7, but will not translate paths to wide character strings.

Chapter 5. Building Berkeley DB for Windows Mobile

Building for Windows Mobile

This page contains general instructions on building Berkeley DB for Windows Mobile platforms using specific compilers.

The `build_wince` directory in the Berkeley DB distribution contains project files for Microsoft Visual 2005 with the Mobile SDK installed:

Project File	Description
Berkeley_DB.sln	Visual Studio 2005 solution
*.vcproj	Visual Studio 2005 project files

These project files can be used to build Berkeley DB for the Windows Mobile platform.

Building Berkeley DB for Windows Mobile

Visual Studio 2005

1. Choose *File* -> *Open Workspace*.... Navigate to the `build_wince` directory, select `Berkeley_DB` and click `Open`.
2. Select the desired target platform from the platform drop-down menu.
3. Build the desired projects.

Build results

The results of your build will be placed in any one of the following Berkeley DB subdirectories, depending on the configuration that you chose:

```
build_wince\(Platform)\Debug
build_wince\(Platform)\Release
build_wince\(Platform)\Debug_static
build_wince\(Platform)\Release_static
```

When building your application during development, you must link against `libdb_small51sd.lib` or against `libdb_sql51sd.lib` for SQL features. You can also build using a release version of the Berkeley DB libraries and tools, which will be placed in the `build_wince\(Platform)\Release_static` directory. You must add the `build_wince` directory to the list of include directories of your application's project, or copy the Berkeley DB include files to a location in your Visual Studio include path.

Changing Build Configuration Type

This section contains information on how to change between a dynamic library (.dll) and static library (.lib). The library projects and their default output and configuration in the Release build is as follows:

Project	Default Output	Default Configuration
db_small_static	libdb_small51s.lib	Static Library
db_static	libdb51s.lib	Static Library
db_sql	libdb_sql51.dll	Dynamic Library
db_sql_static	libdb_sql51s.lib	Static Library

To change a project configuration type in Visual Studio 2005, select a project and do the following:

1. Choose *Project->Properties* and navigate to `Configuration Properties`.
2. Under `Project Defaults`, change the `Configuration Type` to your desired type.

Note: After this change, the output file names change to the Visual Studio 2005 defaults based on the project name.

Building Berkeley DB for different target platforms

There are many possible target CPU architectures for a Windows Mobile application. This section outlines the process required to add a new target architecture to the project files supplied with Berkeley DB.

The Visual Studio 2005 project files will by default build for Pocket PC 2003 and Smartphone 2003, and Windows Mobile 6.5.3 Professional. If you want to build for other platforms such as Windows Mobile 6.0, 6.1, or 6.5, you need to follow the steps provided in this section.

Different target architectures are available in different Platform SDK or DTK downloads from Microsoft. The appropriate SDK must be installed for your mobile architecture before you can build for that platform. You can find the downloads at the Microsoft Developer Center page.

Visual Studio 2005

1. Choose *File -> Open Workspace*.... Navigate to the `build_wince` directory, select `Berkeley_DB` and click Open.
2. From the `Solution explorer` window, right-click the `Solution Berkeley_DB` and select `Configuration manager...`
3. In the `Active solution platform:` drop down box select `New...`
4. From the `Type or select the new platform` drop-down box, select a configuration from the ones available and click `OK`.
5. Click `Close` from the `Configuration Manager` dialog box.
6. The target platform drop-down now contains the platform just added.
7. Build as per the instructions given at the beginning of this chapter.

BDB SQL Notes

After building for a different platform, change the deployment remote directory for the `db_sql` project to run the `wce_sql` sample application:

1. Select `db_sql` from the `Solution explorer` window.
2. Choose *Project->Properties* and navigate to `Configuration Properties`.
3. Under `Deployment`, change the remote directory to `%CSIDL_PROGRAM_FILES%\wce_sql`.

Windows Mobile notes

1. The C++ API is not supported on Windows Mobile. The file stream and exception handling functionality provided by the Berkeley DB C++ API are not supported by Windows Mobile. It is possible to build a C++ application against the Berkeley DB C API.
2. The Java API is not currently supported on Windows Mobile.
3. Tcl support is not currently supported on Windows Mobile.
4. Berkeley DB is shipped with support for the Pocket PC 2003 and Smartphone 2003 target platforms. It is possible to build Berkeley DB for different target platforms using Visual Studio's Configuration Manager.
   This can be done using the following steps:
   a. Open Visual Studio, and load the `build_wince/Berkeley_DB.sln` solution file.
   b. Select the *Build->Configuration Manager...* menu item.
   c. In the *Active Solution Platform...* dropdown, select *New...*
   d. Select the desired target platform (you must have the desired Microsoft Platform SDK installed for it to appear in the list). Choose to copy settings from either the Pocket PC 2003 or Smartphone 2003 platforms.

   Before building the wce_tpcb sample application for the new platform, you will need to complete the following steps:
   a. Open the project properties page for wce_tpcb. Do this by: Right click *wce_tpcb* in the *Solution Explorer* then select *Properties*
   b. Select *Configuration Properties->Linker->Input*
   c. Remove `secchk.lib` and `crtti.lib` from the *Additional Dependencies* field.

   NOTE: These steps are based on Visual Studio 2005, and might vary slightly depending on which version of Visual Studio being used.

Windows Mobile FAQ

1. **What if my Windows Mobile device does not support SetFilePointer and/or SetEndOfFile?**

You can manually disable the truncate functionality from the build.

Do that by opening the `db-X.X.X/build_wince/db_config.h` file, and change the line that reads

```
#define HAVE_FTRUCATE 1
```

to read

```
#undef HAVE_FTRUNCATE
```

Making this change disables DB->compact() for btree databases.

2. **Why doesn't automatic log archiving work?**

   The Windows Mobile platform does not have a concept of a working directory. This means that the DB_ARCH_REMOVE and DB_ARCH_ABS flags do not work properly within Windows Mobile, because they rely on having a working directory.

   To work around this issue, you can call log_archive with the DB_ARCH_LOG flag, the list of returned file handles will not contain absolute paths. Your application can take this list of files, construct absolute paths, and delete the files.

3. **Does Berkeley DB support Windows Mobile?**

   Yes.

   Berkeley DB relies on a subset of the Windows API, and some standard C library APIs. These are provided by Windows CE. Windows Mobile is built "on top" of Windows CE.

4. **Does Berkeley DB support Windows CE?**

   Yes.

   Berkeley DB relies on a subset of the Windows API, and some standard C library APIs. These are provided by Windows CE.

5. **What platforms are the supplied sample applications designed for?**

   The supplied sample applications were developed for the Pocket PC 2003 emulator. They are known to work on real pocket PC devices and later versions of the emulator as well.

   The supplied applications are not designed to work with Smartphone devices. The screen size and input mechanisms are not compatible.

6. **I see a file mapping error when opening a Berkeley DB environment or database. What is wrong?**

   The default behavior of Berkeley DB is to use memory mapped files in the environment. Windows Mobile does not allow memory mapped files to be created on flash storage.

   There are two workarounds:

a. Configure the Berkeley DB environment not to use memory mapped files. The options are discussed in detail in Shared memory region.

b. Create the Berkeley DB environment on non-flash storage. It is possible to store database and log files in a different location to using the DB_ENV->set_data_dir() and DB_ENV->set_lg_dir() APIs.

Chapter 6. Building Berkeley DB for UNIX/POSIX

Building for UNIX/POSIX

The Berkeley DB distribution builds up to four separate libraries: the base C API Berkeley DB library and the optional C++, Java, and Tcl API libraries. For portability reasons, each library is standalone and contains the full Berkeley DB support necessary to build applications; that is, the C++ API Berkeley DB library does not require any other Berkeley DB libraries to build and run C++ applications.

Building for Linux, iPhone OS, Mac OS X and the QNX Neutrino release is the same as building for a conventional UNIX platform.

The Berkeley DB distribution uses the Free Software Foundation's autoconf and libtool tools to build on UNIX platforms. In general, the standard configuration and installation options for these tools apply to the Berkeley DB distribution.

To perform a standard UNIX build of Berkeley DB, change to the **build_unix** directory and then enter the following two commands:

```
../dist/configure
make
```

This will build the Berkeley DB library.

To install the Berkeley DB library, enter the following command:

```
make install
```

To rebuild Berkeley DB, enter:

```
make clean
make
```

If you change your mind about how Berkeley DB is to be configured, you must start from scratch by entering the following command:

```
make realclean
../dist/configure
make
```

To uninstall Berkeley DB, enter:

```
make uninstall
```

To build multiple UNIX versions of Berkeley DB in the same source tree, create a new directory at the same level as the build_unix directory, and then configure and build in that directory as described previously.

Building the Berkeley DB SQL Interface

To perform a standard UNIX build of the Berkeley DB SQL interface, go to the **build_unix** directory and then enter the following two commands:

```
../dist/configure --enable-sql
make
```

This creates a library, `libdb_sql`, and a command line tool, `dbsql`. You can create and manipulate SQL databases using the `dbsql` shell.

You can optionally provide the `--enable-sql_compat` argument to the `configure` script. In addition to creating `libdb_sql` and `dbsql` this causes a thin wrapper library called `libsqlite3` and a command line tool called `sqlite3` to be built. This library can be used as a drop-in replacement for SQLite. The `sqlite3` command line tool is identical to the `dbsql` executable but is named so that existing scripts for SQLite can easily work with Berkeley DB.

```
../dist/configure --enable-sql_compat
make
```

There are several arguments you can specify when configuring the Berkeley DB SQL Interface. See Configuring the SQL Interface (page 44) for more information.

Configuring Berkeley DB

There are several arguments you can specify when configuring Berkeley DB. Although only the Berkeley DB-specific ones are described here, most of the standard GNU autoconf arguments are available and supported. To see a complete list of possible arguments, specify the --help flag to the configure program.

The Berkeley DB specific arguments are as follows:

- **--disable-largefile**

  Some systems, notably versions of HP/UX and Solaris, require special compile-time options in order to create files larger than 2^32 bytes. These options are automatically enabled when Berkeley DB is compiled. For this reason, binaries built on current versions of these systems may not run on earlier versions of the system because the library and system calls necessary for large files are not available. To disable building with these compile-time options, enter --disable-largefile as an argument to configure.

- **--disable-shared, --disable-static**

  On systems supporting shared libraries, Berkeley DB builds both static and shared libraries by default. (Shared libraries are built using the GNU Project's Libtool distribution, which supports shared library builds on many (although not all) systems.) To not build shared libraries, configure using the --disable-shared argument. To not build static libraries, configure using the --disable-static argument.

- **--enable-compat185**

  To compile or load Berkeley DB 1.85 applications against this release of the Berkeley DB library, enter --enable-compat185 as an argument to configure. This will include Berkeley DB 1.85 API compatibility code in the library.

- **--enable-cxx**

To build the Berkeley DB C++ API, enter --enable-cxx as an argument to configure.

- **--enable-debug**

  To build Berkeley DB with **-g** as a compiler flag and with **DEBUG** #defined during compilation, enter --enable-debug as an argument to configure. This will create a Berkeley DB library and utilities with debugging symbols, as well as load various routines that can be called from a debugger to display pages, cursor queues, and so forth. If installed, the utilities will not be stripped. This argument should not be specified when configuring to build production binaries.

- **--enable-debug_rop**

  To build Berkeley DB to output log records for read operations, enter --enable-debug_rop as an argument to configure. This argument should not be specified when configuring to build production binaries.

- **--enable-debug_wop**

  To build Berkeley DB to output log records for write operations, enter --enable-debug_wop as an argument to configure. This argument should not be specified when configuring to build production binaries.

- **--enable-diagnostic**

  To build Berkeley DB with run-time debugging checks, enter --enable-diagnostic as an argument to configure. This causes a number of additional checks to be performed when Berkeley DB is running, and also causes some failures to trigger process abort rather than returning errors to the application. Applications built using this argument should not share database environments with applications built without this argument. This argument should not be specified when configuring to build production binaries.

- **--enable-dump185**

  To convert Berkeley DB 1.85 (or earlier) databases to this release of Berkeley DB, enter --enable-dump185 as an argument to configure. This will build the db_dump185 utility, which can dump Berkeley DB 1.85 and 1.86 databases in a format readable by the Berkeley DB db_load utility.

  The system libraries with which you are loading the db_dump185 utility must already contain the Berkeley DB 1.85 library routines for this to work because the Berkeley DB distribution does not include them. If you are using a non-standard library for the Berkeley DB 1.85 library routines, you will have to change the Makefile that the configuration step creates to load the db_dump185 utility with that library.

- **--enable-java**

  To build the Berkeley DB Java API, enter --enable-java as an argument to configure. To build Java, you must also build with shared libraries. Before configuring, you must set your PATH environment variable to include javac. Note that it is not sufficient to include a symbolic link to javac in your PATH because the configuration process uses the location of javac to

determine the location of the Java include files (for example, jni.h). On some systems, additional include directories may be needed to process jni.h; see Changing compile or load options (page 49) for more information.

- **--enable-posixmutexes**

  To force Berkeley DB to use the POSIX pthread mutex interfaces for underlying mutex support, enter --enable-posixmutexes as an argument to configure. This is rarely necessary: POSIX mutexes will be selected automatically on systems where they are the preferred implementation.

  The --enable-posixmutexes configuration argument is normally used in two ways: First, when there are multiple mutex implementations available and the POSIX mutex implementation is not the preferred one (for example, on Solaris where the LWP mutexes are used by default). Second, by default the Berkeley DB library will only select the POSIX mutex implementation if it supports mutexes shared between multiple processes, as described for the pthread_condattr_setpshared and pthread_mutexattr_setpshared interfaces. The --enable-posixmutexes configuration argument can be used to force the selection of POSIX mutexes in this case, which can improve application performance significantly when the alternative mutex implementation is a non-blocking one (for example test-and-set assembly instructions). However, configuring to use POSIX mutexes when the implementation does not have inter-process support will only allow the creation of private database environments, that is, environments where the DB_PRIVATE flag is specified to the DB_ENV->open() method.

  Specifying the --enable-posixmutexes configuration argument may require that applications and Berkeley DB be linked with the -lpthread library.

- **--enable-pthread_api**

  To configure Berkeley DB for a POSIX pthreads application (with the exception that POSIX pthread mutexes may not be selected as the underlying mutex implementation for the build), enter --enable-pthread_api as an argument to configure. The build will include the Berkeley DB replication manager interfaces and will use the POSIX standard pthread_self and pthread_yield functions to identify threads of control and yield the processor. The --enable-pthread_api argument requires POSIX pthread support already be installed on your system.

  Specifying the --enable-pthread_api configuration argument may require that applications and Berkeley DB be linked with the -lpthread library.

- **--enable-sql**

  To build the command tool dbsql, enter --enable-sql as an argument to configure. The dbsql utility provides access to the Berkeley DB SQL interface. See Configuring the SQL Interface (page 44) for more information.

- **--enable-sql_compat**

  To build the command tool sqlite3, enter --enable-sql_compat as an argument to configure. Sqlite3 is a command line tool that enables you to manually enter and execute SQL

commands. It is identical to the dbsql executable but named so that existing scripts for SQLite can easily work with Berkeley DB. See Configuring the SQL Interface (page 44) for more information.

- **--enable-sql_codegen**

  To build the command line tool db_sql_codegen, enter --enable-sql_codegen as an argument to configure. The db_sql_codegen utility translates a schema description written in a SQL Data Definition Language dialect into C code that implements the schema using Berkeley DB.

- **--enable-smallbuild**

  To build a small memory footprint version of the Berkeley DB library, enter --enable-smallbuild as an argument to configure. The --enable-smallbuild argument is equivalent to individually specifying --with-cryptography=no, --disable-hash, --disable-queue, --disable-replication, --disable-statistics and --disable-verify, turning off cryptography support, the Hash and Queue access methods, database environment replication support and database and log verification support. See Building a small memory footprint library (page 48) for more information.

- **--enable-stl**

  To build the Berkeley DB C++ STL API, enter --enable-stl as an argument to configure. Setting this argument implies that --enable-cxx is set, and the Berkeley DB C++ API will be built too.

  There will be a libdb_stl-X.X.a and libdb_stl-X.X.so built, which are the static and shared library you should link your application with in order to make use of Berkeley DB via its STL API.

  If your compiler is not ISO C++ compliant, the configure may fail with this argument specified because the STL API requires standard C++ template features. In this case, you will need a standard C++ compiler. So far gcc is the best choice, we have tested and found that gcc-3.4.4 and all its newer versions can build the Berkeley DB C++ STL API successfully.

  For information on db_stl supported compilers, see the Portability section in the *Programmer's Reference Guide*.

  And you need to include the STL API header files in your application code. If you are using the Berkeley DB source tree, the header files are in <Berkeley DB Source Root >/stl directory; If you are using the installed version, these header files are in < Berkeley DB Installed Directory>/include, as well as the db.h and db_cxx.h header files.

- **--enable-tcl**

  To build the Berkeley DB Tcl API, enter --enable-tcl as an argument to configure. This configuration argument expects to find Tcl's tclConfig.sh file in the `/usr/local/lib` directory. See the --with-tcl argument for instructions on specifying a non-standard location for the Tcl installation. See Loading Berkeley DB with Tcl for information on sites from which

you can download Tcl and which Tcl versions are compatible with Berkeley DB. To build Tcl, you must also build with shared libraries.

- **--enable-test**

  To build the Berkeley DB test suite, enter --enable-test as an argument to configure. To run the Berkeley DB test suite, you must also build the Tcl API. This argument should not be specified when configuring to build production binaries.

- **--enable-uimutexes**

  To force Berkeley DB to use the UNIX International (UI) mutex interfaces for underlying mutex support, enter --enable-uimutexes as an argument to configure. This is rarely necessary: UI mutexes will be selected automatically on systems where they are the preferred implementation.

  The --enable-uimutexes configuration argument is normally used when there are multiple mutex implementations available and the UI mutex implementation is not the preferred one (for example, on Solaris where the LWP mutexes are used by default).

  Specifying the --enable-uimutexes configuration argument may require that applications and Berkeley DB be linked with the -lthread library.

- **--enable-umrw**

  Rational Software's Purify product and other run-time tools complain about uninitialized reads/writes of structure fields whose only purpose is padding, as well as when heap memory that was never initialized is written to disk. Specify the --enable-umrw argument during configuration to mask these errors. This argument should not be specified when configuring to build production binaries.

- **--enable-dtrace** [**--enable-perfmon-statistics**]

  To build Berkeley DB with performance event monitoring probes add --enable-dtrace to the configuration options. Both native DTrace (on Solaris and Mac OS X) and the Statically Defined Tracing compatibility layer in Linux SystemTap version 1.1 or better are supported. That compatibility package may be called systemtap-sdt-devel; it includes `sys/sdt.h`.

  If --enable-perfmon-statistics is combined with --enable-dtrace then additional probes are defined for the tracking variables from which DB's statistics are obtained. They allow DTrace and SystemTap access to these values when they are updated, are the basis of the statistics as displayed db_stat and the API functions that return statistics.

  The --enable-dtrace option may not be specified at the same time as --disable-statistics.

  For information on Berkeley DB Performance Event Monitoring, see the Performance Event Monitoring section in the *Programmer's Reference Guide*.

- **--with-cryptography**

  To build Berkeley DB with support for cryptography, enter --with-cryptography=yes as an argument to configure.

To build Berkeley DB without support for cryptography, enter --with-cryptography=no as an argument to configure.

To build Berkeley DB with support for cryptography using Intel's Performance Primitive (IPP) library, enter --with-cryptography=ipp as an argument to configure. Additionally, set the following arguments:

-L/path/to/ipp/sharedlib to LDFLAGS

-I/path/to/ipp/include to CPPFLAGS

-lippcpem64t -lpthread to LIBS

An example configuration command for IPP encryption is as follows:

```
../dist/configure -with-cryptography=ipp
        CPPFLAGS="-I/opt/intel/ipp/6.1.3.055/em64t/include"
        LDFLAGS="-L/opt/intel/ipp/6.1.3.055/em64t/sharedlib"
        LIBS="-lippcpem64t -lpthread"
```

See the Intel Documenation for specific instructions on configuring environment variables.

Note: The --with-cryptography=ipp argument works only on Linux.

- **--with-mutex=MUTEX**

  To force Berkeley DB to use a specific mutex implementation, configure with --with-mutex=MUTEX, where MUTEX is the mutex implementation you want. For example, --with-mutex=x86/gcc-assembly will configure Berkeley DB to use the x86 GNU gcc compiler based test-and-set assembly mutexes. This is rarely necessary and should be done only when the default configuration selects the wrong mutex implementation. A list of available mutex implementations can be found in the distribution file `dist/aclocal/mutex.m4`.

- **--with-tcl=DIR**

  To build the Berkeley DB Tcl API, enter --with-tcl=DIR, replacing DIR with the directory in which the Tcl tclConfig.sh file may be found. See Loading Berkeley DB with Tcl for information on sites from which you can download Tcl and which Tcl versions are compatible with Berkeley DB. To build Tcl, you must also build with shared libraries.

- **--with-uniquename=NAME**

  To build Berkeley DB with unique symbol names (in order to avoid conflicts with other application modules or libraries), enter --with-uniquename=NAME, replacing NAME with a string that to be appended to every Berkeley DB symbol. If "=NAME" is not specified, a default value of "_MAJORMINOR" is used, where MAJORMINOR is the major and minor release numbers of the Berkeley DB release. See Building with multiple versions of Berkeley DB (page 8) for more information.

Configuring the SQL Interface

There are a set of options you can provide to **configure** in order to control how the Berkeley DB SQL interface is built. These configuration options include:

--enable-sql

Causes the **dbsql** command line interpreter to be built. Along with **dbsql**, this argument also builds the libdb_sqlXX.{so|la} library, a C API library that mirrors the SQLite C API.

--enable-sql_compat

Causes the **sqlite3** command line tool to be built. This tool is identical to the **dbsql** command line tool, except that it has the same name as the command line tool that comes with standard SQLite.

In addition, the libsqlite3.{so|la} C API library is built if this option is specified. This library is identical to the libdb_sqlXX.{so|la} library that is normally built for Berkeley DB's sql interface, except that it has the same name as the library which is built for standard SQLite.

Warning

Use this compatibility option with *extreme* care. Standard SQLite is used by many programs and utilities on many different platforms. Some platforms, such as Mac OS X, come with standard SQLite built in because default applications for the platform use that library.

Use of this option on platforms where standard SQLite is in production use can cause unexpected runtime errors either for your own application, or for applications and utilities commonly found on the platform, depending on which library is found first in the platform's library search path.

Use this option *only* if you know exactly what you are doing.

This option is provided so that there is an easy upgrade path for legacy SQLite tools and scripts that want to use BDB SQL without rewriting the tool or script. However, data contained in standard SQLite databases must be manually migrated from the old database to your BDB SQL database even if you use this option. See the *Berkeley DB Getting Started with the SQL APIs* guide for information on migrating data from standard SQLite to BDB SQL databases.

Note that in addition to the renamed command line tool and library, this option also causes versions of the command line tool and library to be built that use the normal BDB SQLite names (**dbsql** and libdb_sqlXX.{so|la}).

--enable-test

Cause the Berkeley DB SQL interface test suite to be built. This argument can also be used with either `--enable-sql` or `--enable-sql_compat` to build the SQLite Tcl test runner.

--enable-jdbc

Causes the JDBC driver to be built.

The following configuration options are useful when debugging applications:

--enable-debug
: Builds the Berkeley DB SQL interface with debug symbols.

--enable-diagnostic
: Builds the Berkeley DB SQL interface with run-time debugging checks.

Any arguments that you can provide to the standard SQLite configure script can also be supplied when configuring Berkeley DB SQL interface.

Changing Compile Options

- For Berkeley DB SQL to generate each table in a separate file, rather than as subdatabases in a single file, specify the BDBSQL_FILE_PER_TABLE flag as an argument to the configure script using the standard environment variable, CPPFLAGS. When this option is enabled, the SQL database name is used as a directory name. This directory contains one file for the metadata and one file each for every table created by the SQL API. Note that adding or deleting files from the database directory may corrupt your database. To backup the metadata (schema), make a copy of the `metadata` and `table00001` files from the database directory. Make a new copy whenever the schema is changed.
- For Berkeley DB SQL to set the default page size when you create a database, specify the BDBSQL_DEFAULT_PAGE_SIZE flag as an argument to the configure script using the standard environment variable, CPPFLAGS. The value assigned must be a 0, 512, 1024, 2048, 4096, 8192 16384, 32768, or 65536. The default value is 4096. If the value is set to zero, Berkeley DB queries the file system to determine the best page size, and the value of SQLITE_DEFAULT_PAGE_SIZE is used to calculate the cache size, as the cache size is specified as a number of pages.

Enabling Extensions

The Berkeley DB SQL API provides extensions such as full text search and R-Tree index. By default, these two extensions are disabled. To enable an extension in the Berkeley DB SQL interface, specify the related option as an argument to the configure script using the standard environment variable, CPPFLAGS.

SQLITE_ENABLE_FTS3
: Enable building the Berkeley DB full text search layer

SQLITE_ENABLE_RTREE
: Enables the Berkeley DB R-Tree layer.

See the SQLite Documentation for more information on full text search and R-Tree.

Building the JDBC Driver

This section describes how to build the JDBC driver code using `autoconf`, which is the only method supported and tested by the Berkeley DB team.

To build the JDBC driver, you must have Sun Java Development Kit 1.1 or above installed.

```
cd build_unix
../dist/configure --enable-jdbc --prefix=<install path>
```

```
make install
```

You can test the build by entering the following commands from the `build_unix/jdbc` directory:

javac -classpath ./sqlite.jar test3.java
java -Djava.library.path=./.libs -classpath ./sqlite.jar:. test3

Using the JDBC Driver

This section describes the steps to download, build, and run sample programs using the built JDBC driver.

Downloading JDBC Sample Code

The download link for JDBC sample code is available on the Oracle Sun Developer Network (SDN). You can identify the link by the "JDBC programming examples from all three editions (ZIP format)" text beside it.

Modifying Sample Code

Before running the example code, do the following:

1. Unzip the file containing the sample code to a new directory (for example, jdbc_ex).
2. Substitute `jdbc:sqlite:/<db-file-name>` for the generic JDBC URL that appears in the code. That is, put `jdbc:sqlite:/<db-file-name>` between the quotation marks in the line:

   ```
   String url = "jdbc:mySubprotocol:myDataSource";
   ```

   Note: The <db-file-name> can either be an absolute path name like `"/jdbc_ex_db/myDataSource"`, or a relative path-file-name like `"../jdbc_ex_db/myDataSource"`, or a file name, like `"myDataSource"`, in which the database file will be stored at the current directory.
3. Substitute `SQLite.JDBCDriver` for `myDriver.ClassName` in the line: `Class.forName("myDriver.ClassName");`
4. Substitute the username and password you use for your database in the following: `"myLogin", "myPassword"`.

   This is optional.
5. If your JDK version is above 1.5, change the variable name `enum` in `OutputApplet.java` to some other variable name because, as of JDK release 5 `enum` is a keyword and can not be used as an identifier.

Building and Running the JDBC Sample code

See Building the JDBC Driver (page 45) for instructions about building JDBC driver.

To build and run the JDBC examples do the following:

1. Copy `build_unix/jdbc/sqlite.jar` and `build_unix/jdbc/.libs/libsqlite_jni.so` to the `jdbc_ex` directory.

2. In the `jdbc_ex` directory, run the following commands:

```
$ javac -classpath ./sqlite.jar *.java
$ java -classpath .:sqlite.jar -Djava.library.path=. \
<ClassName, eg. CreateCoffees>
```

3. After you run the CreateCoffees example, use the `dbsql` executable to open the `myDataSource` database file and check if the table `COFFEES` has been successfully created in the database.

```
$ dbsql myDataSourcedbsql> .tables
COFFEES
dbsql> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE COFFEES (COF_NAME varchar(32),\
SUP_ID int, PRICE float, SALES int, TOTAL int);
COMMIT;
dbsql>
```

4. Repeat step 3 to run other examples.
   Note: Some examples like AutoGenKeys are not yet supported by BDB JDBC driver. The `SQLFeatureNotSupportedException` is displayed for those unsupported examples.

Building the ODBC Driver

This section describes the steps required to build the ODBC driver.

Configuring Your System

To configure your system prior to building the ODBC driver, do the following:

1. Download and install the latest unixODBC if ODBC is not already installed on your system.

2. Configure the ODBC server to work with SQLite databases. Follow these instructions from Christian Werner.

Building the Library

To build the library, do the following

```
$ cd db-5.1.XX/build_unix
$ CFLAGS="-fPIC" ../dist/configure --enable-sql_compat --disable-shared
$ make
$ cd ..lang/sql/odbc
```

```
$ CFLAGS="-DHAVE_ERRNO_H -I../../../build_unix -I../../../src/dbinc \
         -I../sqlite/src" LDFLAGS="../../../build_unix/libdb-5.1.a" \
         ./configure --with-sqlite3=../generated
$ make
```

The `libsqlite3odbc.so` library containing a statically linked version of Berkeley DB SQL is now built.

NOTE: The final `make` command above is known to generate a warning when using GCC. The warning states: `Warning: Linking the shared library libsqlite3odbc.la against the static library ../../build_unix/libdb-5.1.a is not portable!`. It is generally safe to ignore the warning when using the generated library.

Testing the ODBC Driver

The steps to verify that the installed driver works are as follows:

1. Alter the `/etc/odbcinst.ini` and `~/.odbc.ini` configuration files to refer to the libsqlite3odbc.so file built above.
2. Create a data source, and launch a data source viewer application by doing the following:

   ```
   $ mkdir ~/databases
   $ cd ~/databases
   $ /path/to/Berkeley DB/build_unix/sqlite3 mytest.db
   dbsql> CREATE TABLE t1(x);
   dbsql> .quit;
   $ DataManager
   ```

   The final step opens a GUI application that displays ODBC data sources on a system. You should be able to find the `mytest.db` data source just created.

Building a small memory footprint library

There are a set of configuration options to assist you in building a small memory footprint library. These configuration options turn off specific functionality in the Berkeley DB library, reducing the code size. These configuration options include:

To build Berkeley DB without support for cryptography, enter --with-cryptography=no as an argument to configure.

To build Berkeley DB without support for the Hash access method, enter --disable-hash as an argument to configure.

To build Berkeley DB without support for the Queue access method, enter --disable-queue as an argument to configure.

To build Berkeley DB without support for the database environment replication, enter --disable-replication as an argument to configure.

To build Berkeley DB without support for the statistics interfaces, enter --disable-statistics as an argument to configure.

To build Berkeley DB without support for database verification, enter --disable-verify as an argument to configure.

Equivalent to individually specifying --with-cryptography=no, --disable-hash, --disable-queue, --disable-replication, --disable-statistics and --disable-verify. In addition, when compiling building with the GNU gcc compiler, the --enable-smallbuild option uses the -Os compiler build flag instead of the default -O3.

Note: --disable-cryptography and --enable-cryptography are deprecated in the Berkeley DB 11gR2 release. --with-cryptography=no does the same as --disable-cryptography and --with-cryptography=yes does the same as --enable-cryptography now.

The following configuration options will increase the size of the Berkeley DB library dramatically and are only useful when debugging applications:

--enable-debug
Build Berkeley DB with symbols for debugging.

--enable-debug_rop
Build Berkeley DB with read-operation logging.

--enable-debug_wop
Build Berkeley DB with write-operation logging.

--enable-diagnostic
Build Berkeley DB with run-time debugging checks.

In addition, static libraries are usually smaller than shared libraries. By default Berkeley DB will build both shared and static libraries. To build only a static library, configure Berkeley DB with the Configuring Berkeley DB (page 38) option.

The size of the Berkeley DB library varies depending on the compiler, machine architecture, and configuration options. As an estimate, production Berkeley DB libraries built with GNU gcc version 3.X compilers have footprints in the range of 400KB to 1.2MB on 32-bit x86 architectures, and in the range of 500KB to 1.4MB on 64-bit x86 architectures.

For assistance in further reducing the size of the Berkeley DB library, or in building small memory footprint libraries on other systems, please contact Berkeley DB support.

Changing compile or load options

You can specify compiler and/or compile and load time flags by using environment variables during Berkeley DB configuration. For example, if you want to use a specific compiler, specify the CC environment variable before running configure:

```
prompt: env CC=gcc ../dist/configure
```

Using anything other than the native compiler will almost certainly mean that you'll want to check the flags specified to the compiler and loader, too.

To specify debugging and optimization options for the C compiler, use the CFLAGS environment variable:

```
prompt: env CFLAGS=-O2 ../dist/configure
```

To specify header file search directories and other miscellaneous options for the C preprocessor and compiler, use the CPPFLAGS environment variable:

```
prompt: env CPPFLAGS=-I/usr/contrib/include ../dist/configure
```

To specify debugging and optimization options for the C++ compiler, use the CXXFLAGS environment variable:

```
prompt: env CXXFLAGS=-Woverloaded-virtual ../dist/configure
```

To specify miscellaneous options or additional library directories for the linker, use the LDFLAGS environment variable:

```
prompt: env LDFLAGS="-N32 -L/usr/local/lib" ../dist/configure
```

If you want to specify additional libraries, set the LIBS environment variable before running configure. For example, the following would specify two additional libraries to load, "posix" and "socket":

```
prompt: env LIBS="-lposix -lsocket" ../dist/configure
```

Make sure that you prepend -L to any library directory names and that you prepend -I to any include file directory names! Also, if the arguments you specify contain blank or tab characters, be sure to quote them as shown previously; that is with single or double quotes around the values you are specifying for LIBS.

The env command, which is available on most systems, simply sets one or more environment variables before running a command. If the env command is not available to you, you can set the environment variables in your shell before running configure. For example, in sh or ksh, you could do the following:

```
prompt: LIBS="-lposix -lsocket" ../dist/configure
```

In csh or tcsh, you could do the following:

```
prompt: setenv LIBS "-lposix -lsocket"
prompt: ../dist/configure
```

See your command shell's manual page for further information.

Installing Berkeley DB

Berkeley DB installs the following files into the following locations, with the following default values:

Configuration Variables	Default value
--prefix	/usr/local/BerkeleyDB.**Major**.**Minor**
--exec_prefix	$(prefix)
--bindir	$(exec_prefix)/bin
--includedir	$(prefix)/include
--libdir	$(exec_prefix)/lib
docdir	$(prefix)/docs

Files	Default location
include files	$(includedir)
libraries	$(libdir)
utilities	$(bindir)
documentation	$(docdir)

With one exception, this follows the GNU Autoconf and GNU Coding Standards installation guidelines; please see that documentation for more information and rationale.

The single exception is the Berkeley DB documentation. The Berkeley DB documentation is provided in HTML format, not in UNIX-style man or GNU info format. For this reason, Berkeley DB configuration does not support **--infodir** or **--mandir**. To change the default installation location for the Berkeley DB documentation, modify the Makefile variable, **docdir**.

When installing Berkeley DB on filesystems shared by machines of different architectures, please note that although Berkeley DB include files are installed based on the value of $(prefix), rather than $(exec_prefix), the Berkeley DB include files are not always architecture independent.

To move the entire installation tree to somewhere besides **/usr/local**, change the value of **prefix**.

To move the binaries and libraries to a different location, change the value of **exec_prefix**. The values of **includedir** and **libdir** may be similarly changed.

Any of these values except for **docdir** may be set as part of the configuration:

```
prompt: ../dist/configure --bindir=/usr/local/bin
```

Any of these values, including **docdir**, may be changed when doing the install itself:

```
prompt: make prefix=/usr/contrib/bdb install
```

The Berkeley DB installation process will attempt to create any directories that do not already exist on the system.

Dynamic shared libraries

Warning: the following information is intended to be generic and is likely to be correct for most UNIX systems. Unfortunately, dynamic shared libraries are not standard between UNIX systems, so there may be information here that is not correct for your system. If you have problems, consult your compiler and linker manual pages, or your system administrator.

The Berkeley DB dynamic shared libraries are created with the name libdb-**major**.**minor**.so, where **major** is the major version number and **minor** is the minor version number. Other shared libraries are created if Java and Tcl support are enabled: specifically, libdb_java-**major**.**minor**.so and libdb_tcl-**major**.**minor**.so.

On most UNIX systems, when any shared library is created, the linker stamps it with a "SONAME". In the case of Berkeley DB, the SONAME is libdb-**major**.**minor**.so. It is important to realize that applications linked against a shared library remember the SONAMEs of the libraries they use and not the underlying names in the filesystem.

When the Berkeley DB shared library is installed, links are created in the install lib directory so that libdb-**major**.**minor**.so, libdb-**major**.so, and libdb.so all refer to the same library. This library will have an SONAME of libdb-**major**.**minor**.so.

Any previous versions of the Berkeley DB libraries that are present in the install directory (such as libdb-2.7.so or libdb-2.so) are left unchanged. (Removing or moving old shared

libraries is one drastic way to identify applications that have been linked against those vintage releases.)

Once you have installed the Berkeley DB libraries, unless they are installed in a directory where the linker normally looks for shared libraries, you will need to specify the installation directory as part of compiling and linking against Berkeley DB. Consult your system manuals or system administrator for ways to specify a shared library directory when compiling and linking applications with the Berkeley DB libraries. Many systems support environment variables (for example, LD_LIBRARY_PATH or LD_RUN_PATH), or system configuration files (for example, /etc/ld.so.conf) for this purpose.

Warning: some UNIX installations may have an already existing `/usr/lib/libdb.so`, and this library may be an incompatible version of Berkeley DB.

We recommend that applications link against libdb.so (for example, using -ldb). Even though the linker uses the file named libdb.so, the executable file for the application remembers the library's SONAME (libdb-**major**.**minor**.so). This has the effect of marking the applications with the versions they need at link time. Because applications locate their needed SONAMEs when they are executed, all previously linked applications will continue to run using the library they were linked with, even when a new version of Berkeley DB is installed and the file `libdb.so` is replaced with a new version.

Applications that know they are using features specific to a particular Berkeley DB release can be linked to that release. For example, an application wanting to link to Berkeley DB major release "3" can link using -ldb-3, and applications that know about a particular minor release number can specify both major and minor release numbers; for example, -ldb-3.5.

If you want to link with Berkeley DB before performing library installation, the "make" command will have created a shared library object in the `.libs` subdirectory of the build directory, such as `build_unix/.libs/libdb-major.minor.so`. If you want to link a file against this library, with, for example, a major number of "3" and a minor number of "5", you should be able to do something like the following:

```
cc -L BUILD_DIRECTORY/.libs -o testprog testprog.o -ldb-3.5
env LD_LIBRARY_PATH="BUILD_DIRECTORY/.libs:$LD_LIBRARY_PATH" ./testprog
```

where **BUILD_DIRECTORY** is the full directory path to the directory where you built Berkeley DB.

The libtool program (which is configured in the build directory) can be used to set the shared library path and run a program. For example, the following runs the gdb debugger on the db_dump utility after setting the appropriate paths:

```
libtool gdb db_dump
```

Libtool may not know what to do with arbitrary commands (it is hardwired to recognize "gdb" and some other commands). If it complains the mode argument will usually resolve the problem:

```
libtool --mode=execute my_debugger db_dump
```

On most systems, using libtool in this way is exactly equivalent to setting the LD_LIBRARY_PATH environment variable and then executing the program. On other systems,

using libtool has the virtue of knowing about any other details on systems that don't behave in this typical way.

Running the test suite under UNIX

The Berkeley DB test suite is built if you specify --enable-test as an argument when configuring Berkeley DB. The test suite also requires that you configure and build the Tcl interface to the library.

Before running the tests for the first time, you may need to edit the `include.tcl` file in your build directory. The Berkeley DB configuration assumes that you intend to use the version of the tclsh utility included in the Tcl installation with which Berkeley DB was configured to run the test suite, and further assumes that the test suite will be run with the libraries prebuilt in the Berkeley DB build directory. If either of these assumptions are incorrect, you will need to edit the `include.tcl` file and change the following line to correctly specify the full path to the version of tclsh with which you are going to run the test suite:

```
set tclsh_path ...
```

You may also need to change the following line to correctly specify the path from the directory where you are running the test suite to the location of the Berkeley DB Tcl library you built:

```
set test_path ...
```

It may not be necessary that this be a full path if you have configured your system's shared library mechanisms to search the directory where you built or installed the Tcl library.

All Berkeley DB tests are run from within **tclsh**. After starting tclsh, you must source the file `test.tcl` in the test directory. For example, if you built in the `build_unix` directory of the distribution, this would be done using the following command:

```
% source ../test/tcl/test.tcl
```

If no errors occur, you should get a "%" prompt.

You are now ready to run tests in the test suite; see Running the test suite for more information.

Building SQL Test Suite on Unix

The Berkeley DB SQL interface test suite is built if you specify --enable-test and --enable-sql as arguments, when configuring Berkeley DB. The test suite also requires that you build the Berkeley DB Tcl API.

```
../dist/configure --enable-sql --enable-test --with-tcl=/usr/lib
```

This builds the *testfixture* project in `../build_unix/sql`.

To enable extensions like full text search layer and R-Tree layer in the SQL test suite, configure with --enable-amalgamation.

Architecture independent FAQ

1. **I have gcc installed, but configure fails to find it.**

Berkeley DB defaults to using the native C compiler if none is specified. That is usually "cc", but some platforms require a different compiler to build multithreaded code. To configure Berkeley DB to build with gcc, run configure as follows:

```
env CC=gcc ../dist/configure ...
```

2. **When compiling with gcc, I get unreferenced symbols; for example the following:**

```
symbol __muldi3: referenced symbol not found
symbol __cmpdi2: referenced symbol not found
```

Berkeley DB often uses 64-bit integral types on systems supporting large files, and gcc performs operations on those types by calling library functions. These unreferenced symbol errors are usually caused by linking an application by calling "ld" rather than by calling "gcc": gcc will link in libgcc.a and will resolve the symbols. If that does not help, another possible workaround is to reconfigure Berkeley DB using the --disable-largefile configuration option and then rebuild.

3. **My C++ program traps during a failure in a DB call on my gcc-based system.**

We believe there are some severe bugs in the implementation of exceptions for some gcc compilers. Exceptions require some interaction between compiler, assembler, and runtime libraries. We're not sure exactly what is at fault, but one failing combination is gcc 2.7.2.3 running on SuSE Linux 6.0. The problem on this system can be seen with a rather simple test case of an exception thrown from a shared library and caught in the main program.

A variation of this problem seems to occur on AIX, although we believe it does not necessarily involve shared libraries on that platform.

If you see a trap that occurs when an exception might be thrown by the Berkeley DB runtime, we suggest that you use static libraries instead of shared libraries. See the documentation for configuration. If this doesn't work and you have a choice of compilers, try using a more recent gcc- or a non-gcc based compiler to build Berkeley DB.

Finally, you can disable the use of exceptions in the C++ runtime for Berkeley DB by using the DB_CXX_NO_EXCEPTIONS flag with the DbEnv or Db constructors. When this flag is on, all C++ methods fail by returning an error code rather than throwing an exception.

4. **I get unexpected results and database corruption when running threaded programs.**

I get error messages that mutex (for example, pthread_mutex_XXX or mutex_XXX) functions are undefined when linking applications with Berkeley DB.

On some architectures, the Berkeley DB library uses the ISO POSIX standard pthreads and UNIX International (UI) threads interfaces for underlying mutex support; for example, Solaris and HP-UX. You can specify compilers or compiler flags, or link with the appropriate thread library when loading your application to resolve the undefined references:

```
cc ... -lpthread ...
cc ... -lthread ...
```

```
xlc_r ...
cc ... -mt ...
```

See the appropriate architecture-specific Reference Guide pages for more information.

On systems where more than one type of mutex is available, it may be necessary for applications to use the same threads package from which Berkeley DB draws its mutexes. For example, if Berkeley DB was built to use the POSIX pthreads mutex calls for mutex support, the application may need to be written to use the POSIX pthreads interfaces for its threading model. This is only conjecture at this time, and although we know of no systems that actually have this requirement, it's not unlikely that some exist.

In a few cases, Berkeley DB can be configured to use specific underlying mutex interfaces. You can use the --enable-posixmutexes and --enable-uimutexes configuration options to specify the POSIX and Unix International (UI) threads packages. This should not, however, be necessary in most cases.

In some cases, it is vitally important to make sure that you load the correct library. For example, on Solaris systems, there are POSIX pthread interfaces in the C library, so applications can link Berkeley DB using only C library and not see any undefined symbols. However, the C library POSIX pthread mutex support is insufficient for Berkeley DB, and Berkeley DB cannot detect that fact. Similar errors can arise when applications (for example, tclsh) use dlopen to dynamically load Berkeley DB as a library.

If you are seeing problems in this area after you confirm that you're linking with the correct libraries, there are two other things you can try. First, if your platform supports interlibrary dependencies, we recommend that you change the Berkeley DB Makefile to specify the appropriate threads library when creating the Berkeley DB shared library, as an interlibrary dependency. Second, if your application is using dlopen to dynamically load Berkeley DB, specify the appropriate thread library on the link line when you load the application itself.

5. **I get core dumps when running programs that fork children.**

   Berkeley DB handles should not be shared across process forks, each forked child should acquire its own Berkeley DB handles.

6. **I get reports of uninitialized memory reads and writes when running software analysis tools (for example, Rational Software Corp.'s Purify tool).**

   For performance reasons, Berkeley DB does not write the unused portions of database pages or fill in unused structure fields. To turn off these errors when running software analysis tools, build with the --enable-umrw configuration option.

7. **Berkeley DB programs or the test suite fail unexpectedly.**

   The Berkeley DB architecture does not support placing the shared memory regions on remote filesystems -- for example, the Network File System (NFS) or the Andrew File System (AFS). For this reason, the shared memory regions (normally located in the database home directory) must reside on a local filesystem. See Shared memory region for more information.

With respect to running the test suite, always check to make sure that TESTDIR is not on a remote mounted filesystem.

8. **The db_dump utility fails to build.**

   The db_dump185 utility is the utility that supports the conversion of Berkeley DB 1.85 and earlier databases to current database formats. If the build errors look something like the following, it means the db.h include file being loaded is not a Berkeley DB 1.85 version include file:

```
db_dump185.c: In function `main':
db_dump185.c:210: warning: assignment makes pointer from integer
without a cast
db_dump185.c:212: warning: assignment makes pointer from integer
without a cast
db_dump185.c:227: structure has no member named `seq'
db_dump185.c:227: `R_NEXT' undeclared (first use in this function)
```

   If the build errors look something like the following, it means that the Berkeley DB 1.85 code was not found in the standard libraries:

```
cc -o db_dump185 db_dump185.o
ld:
Unresolved:
dbopen
```

   To build the db_dump185 utility, the Berkeley DB version 1.85 code must already been built and available on the system. If the Berkeley DB 1.85 header file is not found in a standard place, or if the library is not part of the standard libraries used for loading, you will need to edit your Makefile, and change the following lines:

```
DB185INC=
DB185LIB=
```

   So that the system Berkeley DB 1.85 header file and library are found; for example:

```
DB185INC=/usr/local/include
DB185LIB=-ldb185
```

AIX

1. **I can't compile and run multithreaded applications.**

   Special compile-time flags are required when compiling threaded applications on AIX. If you are compiling a threaded application, you must compile with the _THREAD_SAFE flag and load with specific libraries; for example, "-lc_r". Specifying the compiler name with a trailing "_r" usually performs the right actions for the system.

```
xlc_r ...
cc -D_THREAD_SAFE -lc_r ...
```

   The Berkeley DB library will automatically build with the correct options.

2. **I can't run using the DB_SYSTEM_MEM option to DB_ENV->open().**

   AIX 4.1 allows applications to map only 10 system shared memory segments. In AIX 4.3, this has been raised to 256K segments, but only if you set the environment variable "export EXTSHM=ON".

3. **On AIX 4.3.2 (or before) I see duplicate symbol warnings when building the C++ shared library and when linking applications.**

   We are aware of some duplicate symbol warnings with this platform, but they do not appear to affect the correct operation of applications.

4. **On AIX 4.3.3 I see undefined symbols for DbEnv::set_error_stream, Db::set_error_stream or DbEnv::verify when linking C++ applications. (These undefined symbols also appear when building the Berkeley DB C++ example applications).**

   By default, Berkeley DB is built with _LARGE_FILES set to 1 to support the creation of "large" database files. However, this also affects how standard classes, like iostream, are named internally. When building your application, use a "-D_LARGE_FILES=1" compilation option, or insert "#define _LARGE_FILES 1" before any #include statements.

5. **I can't create database files larger than 1GB on AIX.**

   If you're running on AIX 4.1 or earlier, try changing the source code for `os/os_open.c` to always specify the **O_LARGEFILE** flag to the open(2) system call, and recompile Berkeley DB from scratch.

   Also, the documentation for the IBM Visual Age compiler states that it does not not support the 64-bit filesystem APIs necessary for creating large files; the ibmcxx product must be used instead. We have not heard whether the GNU gcc compiler supports the 64-bit APIs or not.

   Finally, to create large files under AIX, the filesystem has to be configured to support large files and the system wide user hard-limit for file sizes has to be greater than 1GB.

6. **I see errors about "open64" when building Berkeley DB applications.**

   System include files (most commonly fcntl.h) in some releases of AIX, HP-UX and Solaris redefine "open" when large-file support is enabled for applications. This causes problems when compiling applications because "open" is a method in the Berkeley DB APIs. To work around this problem:

   a. Avoid including the problematical system include files in source code files which also include Berkeley DB include files and call into the Berkeley DB API.

   b. Before building Berkeley DB, modify the generated include file db.h to itself include the problematical system include files.

   c. Turn off Berkeley DB large-file support by specifying the --disable-largefile configuration option and rebuilding.

7. **I see the error "Redeclaration of lseek64" when building Berkeley DB with the --enable-sql and --enable-test options.**

   In some releases of AIX, the system include files (most commonly `unistd.h`) redefine `lseek` to `lseek64` when large-file support is enabled even though `lseek` may have already been defined when the `_LARGE_FILE_API` macro is on. To work around this problem, do either one of the following:

   a. Disable large-file support in Berkeley DB by specifying the `--disable-largefile` configuration option and rebuilding.

   b. Edit `db.h` manually after running the configure command, and remove the line that includes `unistd.h`.

FreeBSD

1. **I can't compile and run multithreaded applications.**

   Special compile-time flags are required when compiling threaded applications on FreeBSD. If you are compiling a threaded application, you must compile with the _THREAD_SAFE and -pthread flags:

   ```
   cc -D_THREAD_SAFE -pthread ...
   ```

   The Berkeley DB library will automatically build with the correct options.

2. **I see fsync and close system call failures when accessing databases or log files on NFS-mounted filesystems.**

   Some FreeBSD releases are known to return ENOLCK from fsync and close calls on NFS-mounted filesystems, even though the call has succeeded. The Berkeley DB code should be modified to ignore ENOLCK errors, or no Berkeley DB files should be placed on NFS-mounted filesystems on these systems.

HP-UX

1. **I can't specify the DB_SYSTEM_MEM flag to DB_ENV->open().**

   The `shmget`(2) interfaces are not always used on HP-UX, even though they exist, because anonymous memory allocated using `shmget`(2) cannot be used to store the standard HP-UX msemaphore semaphores. For this reason, it may not be possible to specify the DB_SYSTEM_MEM flag on some versions of HP-UX. (We have seen this problem only on HP-UX 10.XX, so the simplest workaround may be to upgrade your HP-UX release.)

2. **I can't specify both the DB_PRIVATE and DB_THREAD flags to DB_ENV->open().**

   It is not possible to store the standard HP-UX msemaphore semaphores in memory returned by `malloc`(3) in some versions of HP-UX. For this reason, it may not be possible to specify both the DB_PRIVATE and DB_THREAD flags on some versions of HP-UX. (We have seen this problem only on some older HP-UX platforms, so the simplest workaround may be to upgrade your HP-UX release.)

3. **I can't compile and run multithreaded applications.**

   Special compile-time flags are required when compiling threaded applications on HP-UX. If you are compiling a threaded application, you must compile with the _REENTRANT flag:

   ```
   cc -D_REENTRANT ...
   ```

   The Berkeley DB library will automatically build with the correct options.

4. **An ENOMEM error is returned from DB_ENV->open() or DB_ENV->remove().**

   Due to the constraints of the PA-RISC memory architecture, HP-UX does not allow a process to map a file into its address space multiple times. For this reason, each Berkeley DB environment may be opened only once by a process on HP-UX; that is, calls to DB_ENV->open() will fail if the specified Berkeley DB environment has been opened and not subsequently closed.

5. **When compiling with gcc, I see the following error:**

   ```
   #error "Large Files (ILP32) not supported in strict ANSI mode."
   ```

   We believe this is an error in the HP-UX include files, but we don't really understand it. The only workaround we have found is to add -D__STDC_EXT__ to the C preprocessor defines as part of compilation.

6. **When using the Tcl or Perl APIs (including running the test suite), I see the error "Can't shl_load() a library containing Thread Local Storage".**

   This problem happens when HP-UX has been configured to use pthread mutex locking, and an attempt is made to call Berkeley DB using the Tcl or Perl APIs. We have never found any way to fix this problem as part of the Berkeley DB build process. To work around the problem, rebuild tclsh or Perl, and modify its build process to explicitly link it against the HP-UX pthread library (currently /usr/lib/libpthread.a).

7. **When running an executable that has been dynamically linked against the Berkeley DB library, I see the error "Can't find path for shared library" even though I correctly set the SHLIB_PATH environment variable.**

   By default, some versions of HP-UX ignore the dynamic library search path specified by the SHLIB_PATH environment variable. To work around this, specify the "+s" flag to ld when linking, or run the following command on the executable that is not working:

   ```
   chatr +s enable -l /full/path/to/libdb-3.2.sl ...
   ```

8. **When building for an IA64 processor, I see either bus errors or compiler warnings about converting between unaligned types (#4232). How can I resolve them?**

   Berkeley DB requires that data types containing members with different sizes be aligned in a consistent way. The HP-UX compiler does not provide this alignment property by default.

   The compiler can be made to generate adequately aligned data by passing the +u1 option to the compiler. See the HP documentation about the +u1 flag for more information.

9. **I see errors about "open64" when building Berkeley DB applications.**

   System include files (most commonly fcntl.h) in some releases of AIX, HP-UX and Solaris redefine "open" when large-file support is enabled for applications. This causes problems when compiling applications because "open" is a method in the Berkeley DB APIs. To work around this problem:

   a. Avoid including the problematical system include files in source code files which also include Berkeley DB include files and call into the Berkeley DB API.

   b. Before building Berkeley DB, modify the generated include file db.h to itself include the problematical system include files.

   c. Turn off Berkeley DB large-file support by specifying the --disable-largefile configuration option and rebuilding.

iPhone OS

Building Berkeley DB in iPhone OS is the same as building for a conventional UNIX platform. This section lists the commands for building Berkeley DB in both the iPhone simulator (a software simulator included in the iPhone SDK that you can use to test your application without using the iPhone/iPod Touch) and the iPhone device.

Prior to building BDB in an iPhone simulator/iPhone device, set the required environment variables for iPhone OS.

The steps to build BDB in an iPhone simulator is as follows:

```
export LDFLAGS="-arch i386 -pipe -Os -gdwarf-2 -no-cpp-precomp"
export CFLAGS="-arch i386 -pipe -Os -gdwarf-2 -no-cpp-precomp "
export CXXFLAGS="-arch i386 -pipe -Os -gdwarf-2 -no-cpp-precomp"
cd $BDB_HOME/build_unix
../dist/configure --host=i386-apple-darwin\
                  --prefix=$SDKROOT ...
make
```

The steps to build BDB in an iPhone device is as follows:

```
export LDFLAGS="-arch armv6 -pipe -Os -gdwarf-2\
                -no-cpp-precomp -mthumb -isysroot $SDKROOT "
export CFLAGS="-arch armv6 -pipe -Os -gdwarf-2\
               -no-cpp-precomp -mthumb -isysroot $SDKROOT "
export CXXFLAGS="-arch armv6 -pipe -Os -gdwarf-2\
                 -no-cpp-precomp -mthumb -isysroot $SDKROOT "
cd $BDB_HOME/build_unix
../dist/configure --host=arm-apple-darwin9\
                  --prefix=$SDKROOT ...
make
```

Both set of commands create the BDB dynamic library - libdb-5.1.dylib. To build the static library, libdb-5.1.a, add the `--enable-shared=no` option while configuring.

IRIX

1. **I can't compile and run multithreaded applications.**

   Special compile-time flags are required when compiling threaded applications on IRIX. If you are compiling a threaded application, you must compile with the _SGI_MP_SOURCE flag:

   ```
   cc -D_SGI_MP_SOURCE ...
   ```

   The Berkeley DB library will automatically build with the correct options.

Linux

1. **I can't compile and run multithreaded applications.**

   Special compile-time flags are required when compiling threaded applications on Linux. If you are compiling a threaded application, you must compile with the _REENTRANT flag:

   ```
   cc -D_REENTRANT ...
   ```

   The Berkeley DB library will automatically build with the correct options.

2. **I see database corruption when accessing databases.**

   Some Linux filesystems do not support POSIX filesystem semantics. Specifically, ext2 and early releases of ReiserFS, and ext3 in some configurations, do not support "ordered data mode" and may insert random data into database or log files when systems crash. Berkeley DB files should not be placed on a filesystem that does not support, or is not configured to support, POSIX semantics.

3. **What scheduler should I use?**

   In some Linux kernels you can select schedulers, and the default is the "anticipatory" scheduler. We recommend not using the "anticipatory" scheduler for transaction processing workloads.

Mac OS X

1. **When trying to link multiple Berkeley DB language interfaces (for example, Tcl, C++, Java, Python) into a single process, I get "multiple definitions" errors from dyld.**

   To fix this problem, set the environment variable MACOSX_DEPLOYMENT_TARGET to 10.3 (or your current version of OS X), and reconfigure and rebuild Berkeley DB from scratch. See the OS X ld(1) and dyld(1) man pages for information about how OS X handles symbol namespaces, as well as undefined and multiply-defined symbols.

2. **When trying to use system-backed shared memory on OS X I see failures about "too many open files".**

The default number of shared memory segments on OS X is too low. To fix this problem, edit the file /etc/rc, changing the kern.sysv.shmmax and kern.sysv.shmseg values as follows:

```
*** /etc/rc.orig        Fri Dec 19 09:34:09 2003
--- /etc/rc     Fri Dec 19 09:33:53 2003
***************
*** 84,93 ****
   # System tuning
   sysctl -w kern.maxvnodes=$(echo $(sysctl -n hw.physmem) '33554432 /
512 * 1024 +p'|dc)
! sysctl -w kern.sysv.shmmax=4194304
   sysctl -w kern.sysv.shmmin=1
   sysctl -w kern.sysv.shmmni=32
! sysctl -w kern.sysv.shmseg=8
   sysctl -w kern.sysv.shmall=1024
   if [ -f /etc/sysctl-macosxserver.conf ]; then
         awk '{ if (!-1 && -1) print $1 }' <
/etc/sysctl-macosxserver.conf | while read
--- 84,93 ----
   # System tuning
   sysctl -w kern.maxvnodes=$(echo $(sysctl -n hw.physmem) '33554432 /
512 * 1024 +p'|dc)
! sysctl -w kern.sysv.shmmax=134217728
   sysctl -w kern.sysv.shmmin=1
   sysctl -w kern.sysv.shmmni=32
! sysctl -w kern.sysv.shmseg=32
   sysctl -w kern.sysv.shmall=1024
   if [ -f /etc/sysctl-macosxserver.conf ]; then
         awk '{ if (!-1 && -1) print $1 }' <
 /etc/sysctl-macosxserver.conf | while read
```

and then reboot the system.

OSF/1

1. **I can't compile and run multithreaded applications.**

   Special compile-time flags are required when compiling threaded applications on OSF/1. If you are compiling a threaded application, you must compile with the _REENTRANT flag:

   ```
   cc -D_REENTRANT ...
   ```

   The Berkeley DB library will automatically build with the correct options.

QNX

1. **To what versions of QNX has DB been ported?**

Berkeley DB has been ported to the QNX Neutrino technology which is commonly referred to as QNX RTP (Real-Time Platform). Berkeley DB has not been ported to earlier versions of QNX, such as QNX 4.25.

2. **Building Berkeley DB shared libraries fails.**

   The `/bin/sh` utility distributed with some QNX releases drops core when running the GNU libtool script (which is used to build Berkeley DB shared libraries). There are two workarounds for this problem: First, only build static libraries. You can disable building shared libraries by specifying the configuration flag when configuring Berkeley DB.

   Second, build Berkeley DB using an alternate shell. QNX distributions include an accessories disk with additional tools. One of the included tools is the GNU bash shell, which is able to run the libtool script. To build Berkeley DB using an alternate shell, move `/bin/sh` aside, link or copy the alternate shell into that location, configure, build and install Berkeley DB, and then replace the original shell utility.

3. **Are there any QNX filesystem issues?**

   Berkeley DB generates temporary files for use in transactionally protected file system operations. Due to the filename length limit of 48 characters in the QNX filesystem, applications that are using transactions should specify a database name that is at most 43 characters.

4. **What are the implications of QNX's requirement to use `shm_open(2)` in order to use `mmap(2)`?**

   QNX requires that files mapped with `mmap`(2) be opened using `shm_open`(2). There are other places in addition to the environment shared memory regions, where Berkeley DB tries to memory map files if it can.

   The memory pool subsystem normally attempts to use `mmap`(2) even when using private memory, as indicated by the DB_PRIVATE flag to DB_ENV->open(). In the case of QNX, if an application is using private memory, Berkeley DB will not attempt to map the memory and will instead use the local cache.

5. **What are the implications of QNX's mutex implementation using microkernel resources?**

   On QNX, the primitives implementing mutexes consume system resources. Therefore, if an application unexpectedly fails, those resources could leak. Berkeley DB solves this problem by always allocating mutexes in the persistent shared memory regions. Then, if an application fails, running recovery or explicitly removing the database environment by calling the DB_ENV->remove() method will allow Berkeley DB to release those previously held mutex resources. If an application specifies the DB_PRIVATE flag (choosing not to use persistent shared memory), and then fails, mutexes allocated in that private memory may leak their underlying system resources. Therefore, the DB_PRIVATE flag should be used with caution on QNX.

6. **The make clean command fails to execute when building the Berkeley DB SQL interface.**

   Remove the build directory manually to clean up and proceed.

SCO

1. **If I build with gcc, programs such as db_dump and db_stat core dump immediately when invoked.**

   We suspect gcc or the runtime loader may have a bug, but we haven't tracked it down. If you want to use gcc, we suggest building static libraries.

Solaris

1. **I can't compile and run multithreaded applications.**

   Special compile-time flags and additional libraries are required when compiling threaded applications on Solaris. If you are compiling a threaded application, you must compile with the D_REENTRANT flag and link with the libpthread.a or libthread.a libraries:

   ```
   cc -mt ...
   cc -D_REENTRANT ... -lthread
   cc -D_REENTRANT ... -lpthread
   ```

   The Berkeley DB library will automatically build with the correct options.

2. **I've installed gcc on my Solaris system, but configuration fails because the compiler doesn't work.**

   On some versions of Solaris, there is a cc executable in the user's path, but all it does is display an error message and fail:

   ```
   % which cc
   /usr/ucb/cc
   % cc
   /usr/ucb/cc: language optional software package not installed
   ```

   Because Berkeley DB always uses the native compiler in preference to gcc, this is a fatal error. If the error message you are seeing is the following, then this may be the problem:

   ```
   checking whether the C compiler (cc -O) works... no
   configure: error: installation or configuration problem: C compiler
   cannot create executables.
   ```

   The simplest workaround is to set your CC environment variable to the system compiler and reconfigure; for example:

   ```
   env CC=gcc ../dist/configure
   ```

   If you are using the --configure-cxx option, you may also want to specify a C++ compiler, for example the following:

```
env CC=gcc CCC=g++ ../dist/configure
```

3. **I see the error "libc internal error: _rmutex_unlock: rmutex not held", followed by a core dump when running threaded or JAVA programs.**

   This is a known bug in Solaris 2.5 and it is fixed by Sun patch 103187-25.

4. **I see error reports of nonexistent files, corrupted metadata pages and core dumps.**

   Solaris 7 contains a bug in the threading libraries (-lpthread, -lthread), which causes the wrong version of the pwrite routine to be linked into the application if the thread library is linked in after the C library. The result will be that the pwrite function is called rather than the pwrite64. To work around the problem, use an explicit link order when creating your application.

   Sun Microsystems is tracking this problem with Bug Id's 4291109 and 4267207, and patch 106980-09 to Solaris 7 fixes the problem:

   ```
   Bug Id: 4291109
   Duplicate of: 4267207
   Category: library
   Subcategory: libthread
   State: closed
   Synopsis: pwrite64 mapped to pwrite
   Description:
   When libthread is linked after libc, there is a table of functions in
   libthread that gets "wired into" libc via _libc_threads_interface().
   The table in libthread is wrong in both Solaris 7 and on28_35 for the
   TI_PWRITE64 row (see near the end).
   ```

5. **I see corrupted databases when doing hot backups or creating a hot failover archive.**

   The Solaris cp utility is implemented using the mmap system call, and so writes are not blocked when it reads database pages. See Berkeley DB recoverability for more information.

6. **Performance is slow and the application is doing a lot of I/O to the disk on which the database environment's files are stored.**

   By default, Solaris periodically flushes dirty blocks from memory-mapped files to the backing filesystem. This includes the Berkeley DB database environment's shared memory regions and can affect Berkeley DB performance. Workarounds include creating the shared regions in system shared memory (DB_SYSTEM_MEM) or application private memory (DB_PRIVATE), or configuring Solaris to not flush memory-mapped pages. For more information, see the "Solaris Tunable Parameters Reference Manual: fsflush and Related Tunables".

7. **I see errors about "open64" when building Berkeley DB applications.**

   System include files (most commonly fcntl.h) in some releases of AIX, HP-UX and Solaris redefine "open" when large-file support is enabled for applications. This causes problems

when compiling applications because "open" is a method in the Berkeley DB APIs. To work around this problem:

a. Avoid including the problematical system include files in source code files which also include Berkeley DB include files and call into the Berkeley DB API.

b. Before building Berkeley DB, modify the generated include file db.h to itself include the problematical system include files.

c. Turn off Berkeley DB large-file support by specifying the --disable-largefile configuration option and rebuilding.

SunOS

1. **I can't specify the DB_SYSTEM_MEM flag to DB_ENV->open().**

   The `shmget`(2) interfaces are not used on SunOS releases prior to 5.0, even though they apparently exist, because the distributed include files did not allow them to be compiled. For this reason, it will not be possible to specify the DB_SYSTEM_MEM flag to those versions of SunOS.

Ultrix

1. **Configuration complains that mmap(2) interfaces aren't being used.**

   The `mmap`(2) interfaces are not used on Ultrix, even though they exist, because they are known to not work correctly.

Chapter 7. Building Berkeley DB for VxWorks

Building for VxWorks 5.4 and 5.5

The build_vxworks directory in the Berkeley DB distribution contains a workspace and project files for Tornado 2.0/VxWorks 5.4 and Tornado 2.2/VxWorks 5.5.

File	Description
BerkeleyDB20.wsp	Berkeley DB Workspace file for Tornado 2.0
BerkeleyDB20.wpj	Berkeley DB Project file for Tornado 2.0
BerkeleyDB22.wsp	Berkeley DB Workspace file for Tornado 2.2
BerkeleyDB22.wpj	Berkeley DB Project file for Tornado 2.2
dbdemo/dbdemo20.wpj	VxWorks notes (page 69) project file for Tornado 2.0
dbdemo/dbdemo22.wpj	VxWorks notes (page 69) project file for Tornado 2.2
db_*/*20.wpj	VxWorks notes (page 69) project files for Tornado 2.0
db_*/*22.wpj	VxWorks notes (page 69) project files for Tornado 2.2

Building With Tornado 2.0 or Tornado 2.2

Open the workspace **BerkeleyDB20.wsp** or **BerkeleyDB22.wsp**. The list of projects in this workspace will be shown. These projects were created for the x86 BSP for VxWorks.

The remainder of this document assumes that you already have a VxWorks target and a target server, both up and running. It also assumes that your VxWorks image is configured properly for your needs. It also assumes that you have an acceptable file system already available. See VxWorks FAQ (page 70) for more information about file system requirements. See VxWorks notes (page 69) for more information about building a small footprint version of Berkeley DB.

First, you need to set the include directories. To do this, go to the *Builds* tab for the workspace. Open up *Berkeley DB Builds*. You will see several different builds, containing different configurations. All of the projects in the Berkeley DB workspace are created to be downloadable applications.

Build	Description
PENTIUM_debug	x86 BSP with debugging
PENTIUM_release	x86 BSP no debugging

You have to add a new build specification if you use a different BSP, want to add a build for the simulator or want to customize further. For instance, if you have the Power PC (PPC) BSP,

you need to add a new build for the PPC tool chain. To do so, select the "Builds" tab, select the Berkeley DB project name, and right-click. Choose the *New Build...* selection and create the new build target. For your new build target, you need to decide whether it should be built for debugging. See the properties of the Pentium builds for ways to configure for each case. After you add this build you, you still need to configure correctly the include directories, as described in the sections that follow.

If you are running with a different BSP, you should remove the build specifications that do not apply to your hardware. We recommend that you do this after you configure any new build specifications first. The Tornado tools will get confused if you have a PENTIUMgnu build specification for a PPC BSP, for instance.

Select the build you are interested in, and right-click. Choose the *Properties...* selection. At this point, a tabbed dialog should appear. In this new window, choose the *C/C++ compiler* tab. In the edit box, you need to modify the full pathname of the *build_vxworks* subdirectory of Berkeley DB, followed by the full pathname of Berkeley DB. Then, click OK. Note that some versions of Tornado (such as the version for Windows) do not correctly handle relative pathnames in the include paths.

To build and download the Berkeley DB downloadable application for the first time requires several steps:

1. Select the build you are interested in, and right-click. Choose the *Set... as Active Build* selection.
2. Select the build you are interested in, and right-click. Choose the *Dependencies...* selection. Run dependencies over all files in the Berkeley DB project.
3. Select the build you are interested in, and right-click. Choose the *Rebuild All (Berkeley DB.out)* selection.
4. Select the Berkeley DB project name, and right-click. Choose the *Download "Berkeley DB.out"* selection.

Note that the output file listed about will really be listed as *BerkeleyDB20.out* or *BerkeleyDB22.out* depending on which version of Tornado you are running. You need to repeat this procedure for all builds you are interested in building, as well as for all of the utility project builds you want to run.

Building for VxWorks 6.x

Building With Wind River Workbench using the Makefile

In VxWorks6.x, developers use Wind River Workbench for software development. Two makefiles are provided in the db/build_vxworks directory. Use them to build Berkeley DB or Berkeley DB small build, using the build chain provided with Wind River Workbench.

We assume that you have installed all necessary VxWorks6.x software including the Wind River Workbench, and that you know how to use it.

Use the following steps to build Berkeley DB:

1. Setting variables in the Makefile. Open the Makefile and modify the BDB_ROOT variable to the path of your Berkeley DB source tree's root directory. You may need to set other variables when you build on different platforms, such as BUILD_SPEC, DEBUG_MODE, PROJECT_TYPE, CC_ARCH_SPEC, VXVER, build tool flags, and BUILD_SPEC specific settings. Please refer to the documentation of the Workbench for a complete list of available values of each variable. You can also find out the list of values by creating a project using the Workbench. Each variable's available values will be listed in the GUI window which assigns values to that variable.

2. Make sure "make" can be found. Basically you need to set the make utility's path to environment variables.

3. Start up the wrenv utility of the Wind River Workbench.

4. In the command console, move to the $(BDB_ROOT)/build_vxworks/ directory, rename the target makefile (Makefile.6x or Makefile.6x.small) to "Makefile", and run "make". The make process will start and create the directory "bdbvxw". It will contain all intermediate object files as well as the final built image "bdbvxw.out".

5. After the "bdbvxw.out" image is built, you can use command tools or the Workbench IDE to download and run it.

6. Test and Verification. There is a dbdemo and test_micro, which you can run to verify whether everything works fine.

VxWorks notes

Berkeley DB currently disallows the DB_TRUNCATE flag to the DB->open() method on VxWorks because the operations this flag represents are not fully supported under VxWorks.

The DB->sync() method is implemented using an ioctl call into the file system driver with the FIOSYNC command. Most, but not all file system drivers support this call. Berkeley DB requires the use of a file system that supports FIOSYNC.

Building and Running the Demo Program

The demo program should be built in a manner very similar to building Berkeley DB. If you want different or additional BSP build specifications you should add them by following the directions indicated in Building for VxWorks 5.4 and 5.5 (page 67).

The demo program can be downloaded and run by calling the entry function **dbdemo** with the pathname of a database to use. The demo program will ask for some input keys. It creates a database and adds those keys into the database, using the reverse of the key as the data value. When complete you can either enter EOF (control-D) or **quit** and the demo program will display all of the key/data items in the database.

Building and Running the Utility Programs

The Berkeley DB utilities can be downloaded and run by calling the function equivalent to the utility's name. The utility functions take a string containing all the supported arguments. The program will then decompose that string into a traditional argc/argv used internally. For example, to execute db_stat utility on a database within an environment you would

execute the following from the windsh prompt. Obviously you would change the pathname and database name to reflect your system.

```
> db_stat "-h /tmp/myenvhome -d mydatabase.db"
```

VxWorks 5.4/5.5: shared memory

The memory on VxWorks is always resident and fully shared among all tasks running on the target. For this reason, the DB_LOCKDOWN flag has no effect and the DB_SYSTEM_MEM flag is implied for any application that does not specify the DB_PRIVATE flag. Note that the DB_SYSTEM_MEM flag requires all applications use a segment ID to ensure the applications do not overwrite each other's database environments: see the DB_ENV->set_shm_key() method for more information.

VxWorks 5.4/5.5: building a small memory footprint library

A default small footprint build is provided. This default provides equivalent to the **--enable-smallbuild** configuration option described in Building a small memory footprint library (page 48). In order to build the small footprint, you should move `db_config.h` aside and copy `db_config_small.h` to `db_config.h`. Then open up the appropriate small workspace file via Tornado and build as usual.

Support for Replication Manager

The Berkeley DB Replication Manager component is available on Vxworks 6.x because it provides support for TCP/IP sockets and POSIX 1003.1 style networking and threads. You must build Berkley DB for Vxworks using the command line. Prior to building Berkeley DB, ensure you set appropriate values for the variables specified in Step 1 of Building for VxWorks 6.x (page 68). To use Berkeley DB Replication Manager, `netLib` and `ioLib` must be present in the Vxworks image.

To use the Berkeley DB on Vxworks 5.x, make the following manual changes.

- Undefine the HAVE_GETADDRINFO, HAVE_REPLICATION_THREADS, and HAVE_SYS_SOCKET_H macros in the Berkeley DB include files `db_config.h` and `db_config_small.h`.
- Remove this line: #include <pthread.h>, present in the Berkeley DB include file `db.h`.

VxWorks FAQ

- **I get the error "Workspace open failed: This project workspace is an older format.", when trying to open the supplied workspace on Tornado 2.0 under Windows.**

  This error will occur if the files were extracted in a manner that adds a CR/LF to lines in the file. Make sure that you download the Berkeley DB ".zip" version of the Berkeley DB distribution, and, when extracting the Berkeley DB sources, that you use an unzipper program that will not do any conversion.

- **I sometimes see spurious output errors about temporary directories.**

  These messages are coming from the `stat`(2) function call in VxWorks. Unlike other systems, there may not be a well known temporary directory on the target. Therefore, we

highly recommend that all applications use DB_ENV->set_tmp_dir() to specify a temporary directory for the application.

- **How can I build Berkeley DB without using Tornado?**

  The simplest way to build Berkeley DB without using Tornado is to configure Berkeley DB on a UNIX system, and then use the Makefile and include files generated by that configuration as the starting point for your build. The Makefile and include files are created during configuration, in the current directory, based on your configuration decisions (for example, debugging vs. non-debugging builds), so you'll need to configure the system for the way you want Berkeley DB to be built.

  Additionally, you'll need to account for the slight difference between the set of source files used in a UNIX build and the set used in a VxWorks build. You can use the following command to create a list of the Berkeley DB VxWorks files. The commands assume you are in the build_vxworks directory of the Berkeley DB distribution:

  ```
  % cat > /tmp/files.sed
  s/<BEGIN> FILE_//
  s/_objects//
  ^D
  % grep FILE_ BerkeleyDB.wpj | grep _objects | sed -f /tmp/files.sed \
  > /tmp/db.files
  ```

  You will then have a template Makefile and include files, and a list of VxWorks-specific source files. You will need to convert this Makefile and list of files into a form that is acceptable to your specific build environment.

- **Does Berkeley DB use floating point registers?**

  Yes, there are a few places in Berkeley DB where floating point computations are performed. As a result, all applications that call *taskSpawn* should specify the **VX_FP_TASK** option.

- **Can I run the test suite under VxWorks?**

  The test suite requires the Berkeley DB Tcl library. In turn, this library requires Tcl 8.4 or greater. In order to run the test suite, you would need to port Tcl 8.4 or greater to VxWorks. The Tcl shell included in *windsh* is not adequate for two reasons. First, it is based on Tcl 8.0. Second, it does not include the necessary Tcl components for adding a Tcl extension.

- **Are all Berkeley DB features available for VxWorks?**

  All Berkeley DB features are available for VxWorks with the exception of the DB_TRUNCATE flag for DB->open(). The underlying mechanism needed for that flag is not available consistently across different file systems for VxWorks.

- **Are there any constraints using particular filesystem drivers?**

  There are constraints using the dosFs filesystems with Berkeley DB. Namely, you must configure your dosFs filesystem to support long filenames if you are using Berkeley DB logging in your application. The VxWorks' dosFs 1.0 filesystem, by default, uses the old MS-

DOS 8.3 file-naming constraints, restricting to 8 character filenames with a 3 character extension. If you have configured with VxWorks' dosFs 2.0 you should be compatible with Windows FAT32 filesystems which supports long filenames.

- **Are there any dependencies on particular filesystem drivers?**

  There is one dependency on specifics of filesystem drivers in the port of Berkeley DB to VxWorks. Berkeley DB synchronizes data using the FIOSYNC function to ioctl() (another option would have been to use the FIOFLUSH function instead). The FIOSYNC function was chosen because the NFS client driver, nfsDrv, only supports it and doesn't support FIOFLUSH. All local file systems, as of VxWorks 5.4, support FIOSYNC -- with the exception of rt11fsLib, which only supports FIOFLUSH. To use rt11fsLib, you will need to modify the os/os_fsync.c file to use the FIOFLUSH function; note that rt11fsLib cannot work with NFS clients.

- **Are there any known filesystem problems?**

  During the course of our internal testing, we came across three problems with the dosFs 2.0 filesystem that warranted patches from Wind River Systems. We strongly recommend you upgrade to dosFs 2.2, **SPR 79795 (x86)** and **SPR 79569 (PPC)** which fixes all of these problems and many more. You should ask Wind River Systems for the patches to these problems if you encounter them and are unable to upgrade to dosFs 2.2.

  The first problem is that files will seem to disappear. You should look at **SPR 31480** in the Wind River Systems' Support pages for a more detailed description of this problem.

  The second problem is a semaphore deadlock within the dosFs filesystem code. Looking at a stack trace via CrossWind, you will see two or more of your application's tasks waiting in semaphore code within dosFs. The patch for this problem is under **SPR 33221** at Wind River Systems. There are several SPR numbers at Wind River Systems that refer to this particular problem.

  The third problem is that all tasks will hang on a dosFs semaphore. You should look at **SPR 72063** in the Wind River Systems' Support pages for a more detailed description of this problem.

- **Are there any filesystems I cannot use?**

  Currently both the Target Server File System (TSFS) and NFS are not able to be used.

  The Target Server File System (TSFS) uses the netDrv driver. This driver does not support any ioctl that allows flushing to the disk, nor does it allow renaming of files via FIORENAME. The NFS file system uses nfsDrv and that driver does not support FIORENAME and cannot be used with Berkeley DB.

- **What VxWorks primitives are used for mutual exclusion in Berkeley DB?**

  Mutexes inside of Berkeley DB use the basic binary semaphores in VxWorks. The mutexes are created using the FIFO queue type.

- **What are the implications of VxWorks' mutex implementation using microkernel resources?**

On VxWorks, the semaphore primitives implementing mutexes consume system resources. Therefore, if an application unexpectedly fails, those resources could leak. Berkeley DB solves this problem by always allocating mutexes in the persistent shared memory regions. Then, if an application fails, running recovery or explicitly removing the database environment by calling the DB_ENV->remove() method will allow Berkeley DB to release those previously held mutex resources. If an application specifies the DB_PRIVATE flag (choosing not to use persistent shared memory), and then fails, mutexes allocated in that private memory may leak their underlying system resources. Therefore, the DB_ENV->open() flag should be used with caution on VxWorks.

Chapter 8. Upgrading Berkeley DB 11.2.5.0 applications to Berkeley DB 11.2.5.1

Introduction

The following pages describe how to upgrade applications coded against the Berkeley DB 11.2.5.0 library to the Berkeley DB 11.2.5.1 library (both of which belong to Berkeley DB 11*g* Release 2). This information does not describe how to upgrade Berkeley DB 1.85 release applications.

For information on the general process of upgrading Berkeley DB installations and upgrade instructions related to historical releases, see the Berkeley DB Upgrade Guide.

DPL Applications must be recompiled

Applications that use the Java interface's *Direct Persistence Layer* must be recompiled, due to a change in the return type of the setter methods in StoreConfig and EvolveConfig classes. The setter methods now return `this` instead of `void`.

Source Tree Rearranged

The source code hierarchy has been reorganized. Source files that belong to Berkeley DB core are now in a top-level directory named `src`. Files related to language interfaces are in `lang`, and all examples are collected under `examples`.

SQLite Interface Upgrade

Berkeley DB's SQL interface includes code from SQLite. The version of SQLite used has been upgraded, so DB SQL is compatible with SQLite version 3.7.0.1. Please see the release notes at http://sqlite.org/changes.html for further information.

Mod_db4 Support Discontinued

The mod_db4 apache module is no longer included in the release.

Berkeley DB Library Version 11.2.5.1 Change Log

This is the changelog for Berkeley DB 11*g* Release 2 (library version 11.2.5.1).

Changes between 11.2.5.1.25 and 11.2.5.1.29

1. Fixed a bug which caused an AccessViolationException when populating a database with a compare delegate specified secondary database. [#18935]

2. Fixed a memory leak in the SQL API that happens when removing an in-memory or temporary database. [#19058]

3. Exclusive transactions will now block new transctions and will prevent existing transactions from making forward progress. [#19256]

4. Fixed a bug that would cause an assert error when opening an in-memory hash database with thread count configured when compiled with --enable-diagnostic. [#19357]

5. Fixed a bug that could cause a hang when deleting a table if there are multiple connections to a database from different processes. [#19419]

6. Fixed a bug in the SQL API when using a blob field with a lot of content and multiple concurrent connections to the database. [#19945]

7. Fixed a bug to correctly handle the DB_BUFFER_SMALL case on a delete operation when compression is enabled. [#19660]

8. Fixed the pragma "txn_priority" so it sets the priority of all SQL transactions in a session. [#19684]

9. Fixed a bug that could cause a crash in replication groups of more than 10 sites, with multiple processes sharing each DB environment concurrently. [#19818]

10. Modified db_hotbackup to not read from the file system as required on non-UNIX systems. Also provided the db_copy function for this purpose. [#19863]

11. Fixed a bug that could cause records in a queue extent to be missing if multiple threads raced to allocate a new page. [#20219]

12. Fixed a bug in the SQL API that could cause a corrupted environment to fail to run recovery automatically. [#20767]

Changes between 11.2.5.1.19 and 11.2.5.1.25

1. Fixed a leak of log file ids when a database is closed before the end of a transaction that references it. [#15957]

2. Fixed a bug in page allocation where if a non-transactional update was being done, then we release the metadata page lock too early possibly leading to the corruption of the in-memory page list used by DB->compact. [#19036]

3. A log write failure on a replication master will now cause a panic since the transaction may be committed on some clients. [#19054]

4. Removed the possibility that checkpoints will overlap in the log, decreasing the time to recover. [#19062]

5. Fixed several bugs that could cause an update running with MVCC to get the wrong version of a page or improperly update the metadata last page number. [#19063]

6. The database open code will no longer log the open and close of the master database in a file when opening a sub database in that file [#19071]

7. Fixed a bug where an error during an update to a hash database with DB_NOOVERWRITE set could return DB_KEYEXIST rather than the correct error. [#19077]

8. Fixed a bug where an updater supporting DB_READ_UNCOMMITED might downgrade its lock too soon if there was an error during the update. [#19155]

9. Fixed a bug that could cause the wrong page number to be on a root or metadata page if DB->compact moved the page and the operation was later rolled forward. [#19167]

10. Fixed a bug that could cause the close of a secondary index database to fail if the transaction doing the open aborted. [#19169]

11. Changed SQL API library build on *nix to link with libpthreads when necessary. [#19098]

12. Fixed a JDBC error where building would fail on Solaris without the flag "-D_HAVE_SQLITE_CONFIG_H". [#19189]

13. Fixed some bugs in log_verify when there are in-memory database logs and subdb logs. [#19157]

14. Fixed a bug that would return DB_NOTFOUND while searching for an existing item using a cursor on a non-sorted duplicate btree database. [#19210]

15. Fixed a JDBC bug that the configure can not deal with CPPFLAGS. [#19234]

16. Fixed a bug in the SQL API that would cause a constraint violation when updating the primary key with the same value. [#18976]

17. Fixed a bug in the SQL BDBSQL_FILE_PER_TABLE option, to allow absolute path names. [#19190]

Changes between 11.2.5.0 and 11.2.5.1.19

Database or Log File On-Disk Format Changes

1. The database file format was unchanged in 11gR2 library version 11.2.5.1.

2. The log file format was unchanged in 11gR2 library version 11.2.5.1.

New Features

1. Added Performance event monitoring support for DTrace and SystemTap which can be enabled during configuration. Static probes have been defined where statistics values are updated, where mutex or transactional consistency lock waits occur, and where some other potentially lengthy operations may be initiated. [#15605]

2. Added a new acknowledge policy - DB_REPMGR_ACKS_ALL_AVAILABLE. [#16762]

3. Added transactional bulk loading optimization for non-nested transactions. [#17669]

4. Added exclusive transaction support for the SQL API. [#17822]

5. Added support for bulk update and delete in C# API. [#18011]

6. Added a db_replicate utility. [#18326]

7. Added an implementation of the Online Backup API. [#18500]

8. Added support in Berkeley DB SQL for the vacuum and incremental vacuum pragmas [#18545]

9. Added an option to automatically convert SQLite databases to Berkeley DB on opening. [#18531]

10. Added BDBSQL_SHARE_PRIVATE, an option to enable inter-process sharing of DB_PRIVATE environments using multiple-reader. [#18533]

11. Added database-level locking to optimize single-threaded operations and remove locking limitations for database load operations. [#18549]

12. Added support for DB_INIT_REP, DB_PRIVATE, DB_REGISTER and DB_THREAD in DB_CONFIG file.[#18555][#18901]

13. Added support for the BDBSQL_DEFAULT_PAGE_SIZE pragma to override Berkeley DB's choice of page size depending on the filesystem. Use SQLITE_DEFAULT_PAGE_SIZE rather than a hard-coded default. [#18577]

14. Added an extension that allows access to binary files stored outside of the database. What is stored in the database is a pointer to the binary file. [#18635]

15. Added .stat command to dbsql shell to print environment, table, and index statistics. [#18640]

16. Added enhancements to reduce the size of indexes in the SQL API by allowing duplicates in the index database and moving the rowid from the index key into the index data. [#18653]

17. Added a compile time flag BDBSQL_FILE_PER_TABLE that causes each table to be created in a separate file. This flag replaces the BDBSQL_SPLIT_META_TABLE flag. [#18664]

18. Added the handling of read only and read write open of the same database in BDB SQL [#18672]

19. Added an encryption implementation to the SQL API [#18683]

Database Environment Changes

1. Fixed failchk behavior on QNX. [#17403]

2. Fixed a bug that prevented the same process from connecting to the database after recovery is performed. [#18535]

3. Fixed a bug which would occur when recovery checkpoint was not written because the cache ran out of space attempting to flush the memory pool cache. The environment would be recovered and all database where made available, but some databases would incorrectly closed. This would cause a subsequent recovery to fail on its backward pass with the error "PANIC: No such file or directory". [#18590]

4. Fixed a bug that could cause recovery to fail with the error "DB_LOGC->get: log record LSN %u/%u: checksum mismatch" if the last log file was nearly full and ended with a

partially written log record which was smaller than a checkpoint record. It now erases the invalid partial record before switching to the new log file. [#18651]

Access Method Changes

1. Fixed a bug such that segementation fault does not occur if DB->set_partition_dirs is called before DB->set_partition. [#18591]

2. Fixed a bug such that the error "unknown path" does not occur if you put duplicate records into a duplicated sorted HASH database with DB_OVERWRITE_DUP option. [#18607]

3. Added the ability to specify that data should not be logged when removing pages from a database. This can be used if the ability to recover the data is not required after the database has been removed. [#18666]

4. Fixed a bug that caused an aborting transaction to fail if it aborted while a DB->compact of the same HASH database was compacting the dynamic hash table [#18695]

5. Fixed a bug that could cause DB->compact to loop on a DB_RECNO database or a database with an multilevel unsorted off page duplicate tree. [#18722]

6. Fixed a bug that could cause an illegal page type error when using a HASH database with MVCC and the HASH table was contracted and then extended. [#18785]

7. Fixed locking bugs: [#18789]

   - The Db->compact method of BTREE with MVCC would return an unpinned page.

   - The RECNO option would fail to lock the next page when splitting a leaf page.

8. Fixed a bug that could cause data to not be returned in a HASH database that was one of multiple databases in a file and was opened prior to running DB->compact method on that database in another thread of control [#18824]

9. Fixed a bug where doing a bulk insert with secondaries could return an error incorrectly. [#18878]

10. Fixed a bug that would return DB_NOTFOUND instead of DB_BUFFER_SMALL when the first item in a HASH database is larger than the user supplied buffer. [#18829]

11. Fixed a bug that would cause handle locks to be left referencing the wrong metadata page if DB->compact moved the metadata page of a sub-database. [#18944]

12. Fixed a bug that could cause records in a queue extent to be missing if multiple threads raced to allocate a new page. [#20219]

API Changes

1. Fixed various items uncovered by extending DB_CONFIG support: [#18720]

   - Added missing set_cache_max method, and fixed name of log_set_config (was set_log_config).

- Added new DB_ENV->repmgr_get_local_site method.
- Fixed a bug which could fail to allocate enough mutexes when specifying a maximum cache size.
- Fixed a bug that could allocate multiple caches when a small cache size was specified.

2. Fixed a leak when aborting a transaction containing a DB->close. [#18951]
3. Fixed a bug that would leave a cursor referencing the wrong hash bucket when the hash table was expanded or contracted. This would cause the new hash bucket not to be locked on subsequent reference but would not cause problems because the metadata page would be locked to the end of the transaction due to the reorganization. [#18986]
4. Fixed a bug that could leave a buffer pinned if a DB->compact operation on HASH had a I/O or deadlock error. [#18980]
5. Fixed a bug that failed to dirty a page when DB->compact moved records within a hash bucket [#18994]
6. Fixed a bug that might cause an update to a HASH database to fail with an "unpinned page returned" error if it first gets an I/O error while logging. [#18985]
7. Fixed a bug that could leave a hash bucket overflow page not linked to the bucket if the unlink of that page aborted. [#19001]
8. Fixed a bug that would leave the next page pointer of a hash bucket that was removed pointing to an invalid page [#19004]

SQL-Specific API Changes

1. Exclusive transactions will now block new transctions and will prevent existing transactions from making forward progress. [#19256]
2. Allowed SQL applications to attach to the same database multiple times unless shared cache mode is explicitly requested. [#18340]
3. Fixed a bug where auto-removal of log files after writing a checkpoint was not functioning correctly. [#18413]
4. Fixed a race between opening and closing SQL databases from multiple threads that could lead to the error "DB_REGISTER limits processes to one open DB_ENV handle per environment". [#18538]
5. Optimized the SQL adapter for joins. Reduce the number of Berkeley DB operations in a join by caching the maximum key in the primary. [#18566]
6. A SQLITE_LOCKED or SQLITE_BUSY error returned by a statement in an explict transaction will no longer invalidate the entire transaction, but just the statement that returned the error. [#18582]

7. Changed how multiple connections to the same database are detected. Used a fileid so that different paths can be used without error. [#18646]

8. Fixed a bug where the journal (environment) directory was being created prior to the actual environment. [#18656]

9. Added a new PRAGMA to allow tuning of when checkpoints are run. [#18657]

10. Fixed spurious "column <x> not unique" error messages.[#18667]

11. Fixed a segmentation fault that could happen when memory could not be allocated for the index key in the SQL API.[#18783]

12. Fixed a bug causing a segfault when releasing a savepoint that was already released. [#18784]

Tcl-Specific API Changes

1. Changed to link tcl8.5 by default on Windows[#18244]

Java-Specific API Changes

1. Fixed a bug where getAllowPopulate and getImmutableSecondaryKey method always returned false for SecondaryConfig objects returned by SecondaryDatabase.getSecondaryConfig method. [#16018]

2. Fixed a bug which made it impossible to (re)set VerboseConfig.REPLICATION_SYSTEM on the Java API. [#17561]

3. Fixed a bug where populating a SecondaryDatabase on open could lead to an OutOfMemoryException. [#18529]

4. Fixed a bug such that segementation fault does not occur when putting records into callback-partitioned database. [#18596]

5. Fixed a bug where DatabaseConfig.getUnsortedDuplicates method returned true when the datbase had been configured for sorted duplicates. [#18612]

6. Initialized DatabaseConfig.pageSize so that it can be queried. [#18691]

7. Fixed a bug by opening a write cursor for Direct Persistent Layer(DPL) entity's put operation in the Concurrent Data Store product. [#18692]

8. Synchronized Java persistence code and tests from Java Edition to Berkeley DB. [#18711]

9. Introduced the EnvironmentConfig.setReplicationInMemory method as a way to configure in-memory internal replication files before opening the Environment handle on the Java API. [#18719]

10. Fixed a bug in the bulk DatabaseEntry class, where it was possible to overflow the buffer. [#18850]

11. Added LEASE_TIMEOUT field to the ReplicationTimeoutType class that enables configuring the amount of time a client grants its Master Lease to a master. [#18867]

C#-Specific API Changes

1. Fixed a bug in BTree prefix comparison method such that there is no problem when the application needs to save a number larger than or equal to 2^31. The BTree prefix comparison function now returns an unsigned int instead of a signed int. [#18481]

2. Fixed a bug which caused the HasMultiple method to throw an exception when there were multiple databases in a single database file. [#18483]

3. Fixed a bug to ensure the CachePriority is set for Database and Cursor objects. [#18716]

4. Fixed a bug that use leading to the error: "Transaction that opened the DB handle is still active" when applications used different transactional handles in the associate and open methods in a secondary database. [#18873]

5. Fixed a bug which caused AccessViolationException when populating db with a compare delegate specified secondary db. [#18935]

Direct Persistence Layer (DPL), Bindings and Collections API

1. All setter methods in the DPL |StoreConfig| and |EvolveConfig| now return |this| rather than having a |void| return type. This change requires that applications using the DPL be recompiled. [#17021]

2. Improve performance of |StoredCollection.removeAll|. This method no longer iterates over all records in the stored collection. [#17727]

3. Several new tuple formats and binding classes have been added in the |com.sleepycat.bind.tuple| package. See the |com.sleepycat.bind.tuple| package description for an overview of the new bindings and a comparative description of all tuple bindings. The new tuple formats are: [#18379]

   - Packed integer formats have been added that support default natural sorting. These are intended to replace the old unsorted packed integer formats.

   - Two new |BigDecimal| formats have been added. One format supports default natural sorting. The other format is unsorted, but has other advantages: trailing zeros after the decimal place are preserved, and a more compact, faster serialization format is used.

4. The following classes are now certified to be serializable. [#18738]

   - com.sleepycat.persist.IndexNotAvailableException

   - com.sleepycat.persist.StoreExistsException

   - com.sleepycat.persist.StoreNotFoundException

   - com.sleepycat.persist.evolve.DeletedClassException

   - com.sleepycat.persist.evolve.IncompatibleClassException

Replication Changes

1. Replication Manager now uses the standard system implementation of getaddrinfo() when running on Windows, which means that it can support IPv6 addresses if support is present and configured in the operating system. [#18263]

2. Fixed a bug which caused a "full election" to fail if a majority of sites were not ready when the election started. [#18456]

3. Fixed a bug which could occur when using bulk transfer with Replication Manager. When closing a DB_ENV handle, any remaining bulk buffer contents are flushed, and Replication Manager could have tried to send the resulting messages even though its connections had already been closed, leading in rare circumstances to spurious EBADF error reports, or possibly even arbitrary memory corruption. [#18469]

4. Fixed a bug which caused Replication Manager to wait for acknowledgement from client, even if it had failed to send a log record, due to "queue limit exceeded". Replication Manager now returns immediately, with a PERM_FAILED indication, to avoid a pointless delay to the commit() operation. [#18682]

5. Fixed a bug where changes made in one process to Replication Manager configuration values (such as ack policy or ack timeout) were not observed in other processes sharing the same database environment. [#18839]

6. Fixed bugs that could prevent client synchronization from completing due to a failure to request missing log records. [#18849]

7. Fixed a bug where a client that had rolled back transactions to synchronize with a new master, failed to invalidate existing database handles later used for cursor operations based on an explicitly provided transaction. [#18862]

8. Fixed a bug where Replication Manager called for an election after a DUPMASTER event, even when using Master Leases. In such a case it now simply accepts the new (remote) master. [#18864]

9. Fixed a bug which would cause failure if client env attempted to perform sync-up recovery to a point in the log that happened to fall exactly on a log file boundary. [#18907]

Locking Subsystem Changes

1. Moved the wait mutex from the lock structure to the locker structure, reducing the number of mutexes required in the system. [#18685]

Memory Pool Subsystem Changes

1. Fixed a race condition that was causing the error: "Unable to allocate space from the buffer cache". The error can only be triggered when multiple memory pool regions are used and there is a periodic gathering and clearing of statistics. This also fixes a second bug where if you compile without statistics and explicitly set the memoru pool default pagesize, other environment handles to that environment would not see the correct memory pool default pagesize. [#18386]

2. Fixed a bug where the get_cachesize method and the mpool_stat method returned the initial cache size, even if the cache size had been changed. [#18706]

3. Changed memory pool allocation so that the EIO error is returned rather than the ENOMEM error when the memory cannot be allocated because dirty pages cannot be written. [#18740]

Mutex Subsystem Changes

1. Fixed problems with the printed statistics for DB_MUTEX_SHARED latches. The DB_STAT_CLEAR flag (as specified by db_stat -Z) did not clear the counts of the number of times a shared latch either had to wait to get access or was able to get the latch without waiting. Also, the ownership state of a test-and-set latch (not a hybrid one) was always displayed as not owned, even when it was held. [#17585] [#18743]

Transaction Subsystem Changes

1. Fixed bugs that could caused PANIC or DB_RUNRECOVERY errors when the synchronization of the transaction log failed. [#18588]

2. Fix javadoc to note the exception to the rule that a transaction handle may not be accessed after commit() operaton (getCommitToken() is allowed). [#18730]

3. Removed the possibility that checkpoints will overlap in the log, decreasing the time to recover [#19062]

Utility Changes

1. Modified the db_printlog and db_dump -da so that they use the same formatting. The db_logprint utility now uses the message stream. Both db_dump -da and db_printlog accept a -D flag to indicate the numer of bytes of data items to display. You can set this value when calling the DB_ENV->set_data_len method or in the DB_CONFIG. [#18365]

2. Fixed a bug that caused a segmentation violation when using the db_printlog utility. [#18694]

3. Fixed a bug in db_hotbackup that would cause a trap if the -D flag is used and the DB_CONFIG file does not specify a log directory. [#18841]

4. Fixed a bug that would cause verify to call the wrong compare function if there are user defined compare functions used and the database has multilevel off page sorted duplicate trees. [#20284]

5. Modified db_hotbackup to not read from the file system as required on non-UNIX systems. Also provided the db_copy function for this purpose. [#19863]

6. Fixed db_hotbackup so that when -d/-l or -D is not specified, DB_CONFIG is used to determine the locations of the databases and logs in the source environment. [#19994]

Configuration, Documentation, Sample Apps, Portability, and Build Changes

1. Added support and documentation for iPhone OS. [#18223]

2. Fix a bug to make the configuration option --enable-debug work when CFLAGS is set. [#18432]
3. Updated Visual Studio project files to enable ADO.NET support. [#18448]
4. Enhanced the source tree layout making it easier to navigate. [#18492]
5. Fixed the --enable-dbm argument to configure. [#18497]
6. Fixed Visual Studio project files so that they can load into Visual Studio 2010. [#18505]
7. Updated Windows CE build files to be consistent with desktop Windows build files. Added Windows Mobile 6.5.3 Professional as a target platform. [#18516]
8. Added a fix so that an error message is displayed when the 'ar' utility is missing when configured. [#18619]
9. Added tighter integration of JDBC on POSIX/autoconf by including an argument --enable-jdbc to configure. [#18621]
10. Fix build conflicts in log verify with other configurations. [#18658]
11. Upgraded Berkeley DB SQL to SQLite version 3.7.0 [#18857]

Example Changes

1. Renamed examples/c/bench_001 to examples/c/ex_bulk. [#18537]

Miscellaneous Bug Fixes

1. Provided a functionality on the Windows platform to choose a default page size based on the underlying file system sector size. [#16538]
2. Changed DB_NOSYNC from an "operation" constant to a flag value. [#17775]
3. Changed the default permissions for files in the release tree to allow write access. [#17974]
4. Fixed a bug which caused database verification to hang when verifying a database in a Concurrent Data Store environment that performs locking on an environment-wide basis (DB_CDB_ALLDB.) [#18571]
5. Fixed temporary file naming with negative process IDs (e.g., VxWorks). [#18975]

Deprecated Features

1. [#18871] Removed the mod_db4 PHP/Apache wrapper. It only supported Apache 1.3 and has not been actively supported. Use php_db4 instead.

Known Bugs

1. Hybrid shared latches have a race condition which can consume cpu unnecessarily. The bug is triggered when a thread requests exclusive access to a shared latch while it is

locked for reading by other threads. If you run into this situation, contact us for a patch. [#18982]

2. When doing a C# API, 64 bit build, some of the data in the lock statistical field may appear incorrect. This is a data display issue. If you run into this situation, contact us for a patch. [#20769]

Chapter 9. Upgrading Berkeley DB 4.8 applications to Berkeley DB 11.2.5.0

Introduction

The following pages describe how to upgrade applications coded against the Berkeley DB 4.8 release interfaces to the Berkeley DB 11*g* Release 2 interfaces. (Library version 11.2.5.0). This information does not describe how to upgrade Berkeley DB 1.85 release applications.

db_sql Renamed to db_sql_codegen

The db_sql utility is now called db_sql_codegen. This command line utility is not built by default. To build db_sql_codegen, specify `--enable-sql_codegen` when configuring Berkeley DB.

DB_REP_CONF_NOAUTOINIT Replaced

In this release, the `DB_REP_CONF_NOAUTOINIT` flag is replaced by the `DB_REP_CONF_AUTOINIT` flag. This option is ON by default. To turn off automatic internal initialization, call the `DB_ENV->rep_set_config` method with the **which** parameter set to `DB_REP_CONF_AUTOINIT` and the **onoff** parameter set to zero.

Support for Multiple Client-to-Client Peers

A Berkeley DB Replication Manager application can now designate one or more remote sites (called its "peers") to receive client-to-client requests.

In previous releases, there could be only one peer at a time. If you called the `DB_ENV->repmgr_add_remote_site` method specifying site "A" as a peer and you made another call specifying site "B" as a peer, site "B" would become the only peer, and site "A" would no longer be a peer.

Starting with Berkeley DB 11gR2, the same sequence of calls results in both site "A" and site "B" being possible peers. Replication Manager may select any of a site's possible peers to use for client-to-client requests. If the first peer that the Replication Manager selects cannot be used (for example, it is unavailable or it is the current master), Replication Manager attempts to use a different peer if there is more than one peer.

To get the pre-11gR2 peer behavior in this example, you must make an additional call to the `DB_ENV->repmgr_add_remote_site` method, specifying site "A" and a flag value that excludes the `DB_REPMGR_PEER` bit value to remove site "A" as a possible peer.

Cryptography Support

In this release, the configuration options, --disable-cryptography and --enable-cryptography are deprecated. --disable-cryptography is replaced by --with-cryptography=no and --enable-cryptography is replaced by --with-cryptography=yes.

To build Berkeley DB with support for cryptography, enter --with-cryptography=yes as an argument to configure instead of --enable-cryptography.

To build Berkeley DB without support for cryptography, enter --with-cryptography=no as an argument to configure instead of --disable-cryptography.

Berkeley DB now supports encryption using Intel's Performance Primitive (IPP) on Linux. To build Berkeley DB with support for cryptography using Intel's Performance Primitive (IPP) library, enter --with-cryptography=ipp as an argument to configure.

Note: The --with-cryptography=ipp argument works only on Linux.

DB_NOSYNC Flag to Flush Files

Applications must now pass the DB_NOSYNC flag to the methods - `DB->remove`, `DB->rename`, `DB_ENV->dbremove`, and `DB_ENV->dbrename`, to avoid a multi-database file to be flushed from cache. This flag is applicable if you have created the database handle in a non-transactional environment.

By default, all non-transactional database remove/rename operations cause data to be synced to disk. This can now be overridden using the DB_NOSYNC flag so that files can be accessed outside the environment after the database handles are closed.

Dropped Support

Berkeley DB no longer supports Visual Studio 6.0. The earliest version supported is Visual Studio 2005. The build files for Windows Visual Studio 6.0 are removed.

Berkeley DB no longer supports Win9X, Windows Me (Millenium edition), and Windows NT 4.0. The minimum supported windows platform is Windows 2000.

Changing Stack Size

Prior to the 11gR2 release, Berkeley DB limited the stack size for threads it created using the POSIX thread API to 128 KB for 32-bit platforms and 256 KB for 64-bit platforms. In this release, the system default stack size is used unless you run the Berkeley DB configure script with the `--with-stacksize=SIZE` argument to override the default.

Berkeley DB 11g Release 2 Change Log

Changes between 11.2.5.0.26 and 11.2.5.0.32

1. Added Visual Studio 2010 support. Users can find Visual Studio 2010 solutions and projects on build_windows. [#18889]

2. Fixed a leak of log file ids when a database is closed before the end of a transaction that references it. [#15957]

3. Fixed a race condition that was causing an "unable to allocate space from the buffer cache" error. The error can only be triggered when multiple mpool regions are used

and there is a periodic gathering and clearing of statistics. This also fixes a second bug where if you compile without statistics and explicitly set the mpool default pagesize, other environment handles to that environment would not see the correct mpool default pagesize. [#18386]

4. Fix failure to flush pages to disk. [#18760]

5. Fix a general I/O problem on Windows where system doesn't always return ENOENT when file is missing. [#18762]

6. Fixed locking bugs: [#18789]

   - Db->compact of BTREE with MVCC could return an unpinned page.

   - RECNO would fail to lock the next page when splitting a leaf page.

7. Don't await ack if message not sent due to queue limit exceeded. [#18682]

8. Fixed a bug that could cause data to not be returned in a HASH database that was one of multiple databases in a file and it was opened prior to running DB->compact on that database in another thread of control [#18824]

9. Return HANDLE_DEAD on cursor creation that names a specific txn after client callback. [#18862]

10. Remove parting_shot rep_start(CLIENT) in election thread because it can occasionally conflict with rep_start(MASTER) in another thread. [#18946]

11. Fixed a bug that would cause handle locks to be left referencing the wrong metadata page if DB->compact moved the metadata page of a sub-database. [#18944]

12. Fixed a bug that might cause an update to a HASH database to fail with an "unpinned page returned" error if it first gets an I/O error while logging. [#18985]

13. Fixed a bug that failed to dirty a page when DB->compact moved records within a hash bucket [#18994]

14. Fixed a bug in page allocation where if a non-transactional update was being done, then we release the metadata page lock too early possibly leading to the corruption of the in memory page list used by DB->compact. [#19036]

15. A log write failure on a replication master will now cause a panic since the transaction may be committed on some clients. [#19054]

16. Removed the possibility that checkpoints will overlap in the log, decreasing the time to recover [#19062]

17. Fixed a bug that could leave a hash bucket overflow page not linked to the bucket if the unlink of that page aborted. [#19001]

18. Fixed a bug that would leave the next page pointer of a hash bucket that was removed pointing to an invalid page. [#19004]

19. Fixed several bugs that could cause an update running with MVCC to get the wrong version of a page or improperly update the metadata last page number. [#19063]

20. Fixed a bug where an error during an update to a hash database with DB_NOOVERWRITE set could return DB_KEYEXIST rather than the correct error. [#19077]

21. Fixed a bug where an updater supporting DB_READ_UNCOMMITED might downgrade its lock too soon if there was an error during the update [#19155]

22. Fixed a bug that could cause the wrong page number to be on a root or metadata page if DB->compact moved the page and the operation was later rolled forward [#19167]

23. Fixed a bug that could cause the close of a secondary index database to fail if the transaction doing the open aborted [#19169]

24. The database open code will no longer log the open and close of the master database in a file when opening a sub database in that file [#19071]

Changes between 11.2.5.0.21 and 11.2.5.0.26

1. Fixed a bug that might cause recovery to fail if processed part of the log that had previously been recovered and a database which was not present was opened in the log and not closed. [#18459]

2. Fixed a bug which could occur when using bulk transfer with Replication Manager. When closing a DB_ENV handle, any remaining bulk buffer contents are flushed, and Replication Manager could have tried to send the resulting messages even though its connections had already been closed, leading in rare circumstances to spurious EBADF error reports, or possibly even arbitrary memory corruption. [#18469]

3. Fixed a bug in C# HasMultiple() that this function always throws exceptions when there are multiple databases in a single db file. [#18483]

4. Fixed the '--enable-dbm' argument to configure. [#18497]

5. Fixed a bug in the Java API where populating a SecondaryDatabase on open could lead to an OutOfMemoryException. [#18529]

6. Fixed a bug where DB SQL reports "The database disk image is malformed" in "group by" operations. [#18531]

7. Fixed a bug that prevented the same process from reconnecting to the database when DB_REGISTER is being used. [#18535]

8. Fix a race between opening and closing SQL databases from multiple threads that could lead to the error "DB_REGISTER limits processes to one open DB_ENV handle per environment". [#18538]

9. Fixed some bugs that could cause a panic or a DB_RUN_RECOVERY error if the sync of the transaction log failed. [#18588]

10. Fixed a bug which would occur when recovery checkpoint was not written because the cache ran out of space attempting to flush the mpool cache. The environment was

recovered and all database where made available, but some databases were incorrectly closed. This would cause a subsequent recovery to fail on its backward pass with the error "PANIC: No such file or directory". [#18590]

11. Fixed a bug that segementation fault would occur if DB->set_partition_dirs was called before DB->set_partition. [#18591]

12. Fixed a bug that the error of "unknown path" would occur if putting duplicate records to duplicated sorted hash database with DB_OVERWRITE_DUP.[#18607]

13. Fixed a bug where DatabaseConfig.getUnsortedDuplicates() returned true when the datbase had been configured for sorted duplicates. [#18612]

14. Fixed a bug that could cause recovery to fail with the error "DB_LOGC->get: log record LSN %u/%u: checksum mismatch" if the last log file was nearly full and ended with a partially written log record which was smaller than a checkpoint record. It now erases the invalid partial record before switching to the new log file. [#18651]

15. Initialize DatabaseConfig.pageSize so that it can be queried from Java. [#18691]

16. Fixed a bug that might cause an aborting transaction to fail if it aborted while a DB->compact of the same HASH database was compacting the dynamic hash table [#18695]

Changes between 4.8 and 11.2.5.0.21

Database or Log File On-Disk Format Changes

1. The log file format changed in 11.2.5.0.21

New Features

1. Replication Manager sites can specify one or more possible client-to-client peers. [#14776]

2. Added resource management feature in all Berkeley DB APIs to automatically manage cursor and database handles by closing them when they are not required, if they are not yet closed.[#16188]

3. Added a SQL interface to the Berkeley DB library. The interface is based on - and a drop-in-replacement for - the SQLite API. It can be accessed via a command line utility, a C API, or existing APIs built for SQLite. [#16809]

4. Added hash databases support to the DB->compact interface. [#16936]

5. Renamed the "db_sql" utility to "db_sql_codegen". This utility is not built by default. To build this utility, enter --enable-sql_codegen as an argument to configure. [#18265]

6. Added transactional support in db_sql_codegen utility. Specify TRANSACTIONAL or NONTRANSACTIONAL in hint comments in SQL statement, db_sql_codegen enable/disable transaction in generated code accordingly. [#17237]

7. Added the feature read-your-writes consistency that allows client application to check, or wait for a specific transaction to be replicated from the master before reading database. [#17323]

8. Added DB log verification feature, accessible via the API and a new utility. This feature can help debugging and analysis. [#17420]

9. Added support for applications to assign master/client role explicitly at any time. Replication Manager can now be configured not to initiate elections. [#17484]

10. Enhanced the DB->compact method so that it can reassign metadata and root pages from subdatabases to lower numbered pages while compacting a database file that contains multiple databases. This feature helps to free the higher numbered pages and truncate the file. [#17554]

11. Added system diagnostic messages that are ON by default. [#17561]

12. Added the feature to assign a priority level to transactions. When resolving a deadlock:

    - if the transactions have differing priority, the lowest priority transaction is aborted

    - if all transactions have the same priority, the same poilcy that existed before priorities were introduced is used [#17604]

13. Added a feature in which log_archive uses group-wide information for archiving purposes if Replication Manager is in use. [#17664]

14. Added a feature by which the Replication Manager application clients now automatically request any missing information, even when there is no master transaction activity. [#17665]

15. Added support for sharing logs across mixed-endian systems. [#18032]

16. Added an option to specify the first and last pages to the db_dump utility. You can do this by providing -F and -L flags to the db_dump -d option. [#18072]

17. Added Intel Performance Primitive (IPP) AES encryption support. [#18110]

18. Removed support for the configuration option --with-mutex=UNIX/fcntl as of version 4.8. If Berkeley DB was configured to use this type of mutex in an earlier release, switch to a different mutex type or contact Oracle for support. [#18361]

Database Environment Changes

1. Fixed a bug to reflect the correct configuration of the logging subsystem when the DB_ENV->log_set_config method is called with the DB_LOG_ZERO flag in a situation where a DB_ENV handle is open and an environment exists. [#17532]

2. Fixed a bug to prevent memory leak caused when the environment is closed by the named in-memory database in a private database environment which has open named in-memory databases. [#17816]

3. Fixd a race condition in an internal directory-scanning function that returns the ENOENT ("No such file or directory") error, if a file is removed just before a call to stat() or its eqivalent. [#17850]

Access Method Changes

1. Fixed a bug to prevent a page in the hash database from carrying the wrong header information when a group allocation is rolled forward by recovery. [#15414]

2. Improved the sort function for bulk put operations. [#17440]

3. Fixed a bug in the DB->compact method to ensure locking of leaf pages when merging higher level interior nodes or when freeing interior nodes when descending to find a non-zero length key. [#17485][#16466]

4. Fixed a bug to prevent a trap if a cursor is opened or closed when another thread is adjusting cursors due to an update in the same database. [#17602]

5. Fixed a bug that incorrectly lead to the error message "library build did not include support for the Hash access method" [#17672]

6. Fixed a bug to ensure that the DB->exists method accepts the DB_AUTO_COMMIT flag. [#17687]

7. In the past, removing a database from a multi-database file that was opened in an environment always caused dirty pages in the file to be flushed from the cache. In this release, there is no implicit flush as part of a DB->remove for handles opened in an environment. Applications that expect the database file to be flushed will need to add an explicit flush. [#17775]

8. Fixed a bug so that the code does not loop if a DB->compact operation processed a 3 or more level non-sorted off page duplicate tree. [#17831]

9. Fixed a bug that could leave pages pinned in the cache if an allocation failed during a DB->compact operation. [#17845]

10. Fixed a bug to ensure sequences are closed when an EntityStore is closed. [#17951]

11. Fixed a bug that prevented retrieval of a non-duplicate record with DB_GET_BOTH_RANGE in hash sorted duplicate db. In a database configured with sorted duplicate support, when the DBcursor->get method is passed the DB_GET_BOTH_RANGE flag, the data item should be retrieved that is the smallest value greater than or equal to the value provided by the data parameter (as determined by the comparison function). [#17997]

12. Fixed a bug that causes the wrong file to be removed if multiple cascading renames are done in the same transaction. [#18069]

13. Fixed a bug to prevent the DB->compact method specified with the DB_AUTO_COMMIT flag from acquiring too many locks. [#18072]

14. Fixed a bug that might cause DB->compact on a DB_BTREE database to get a spurious read lock on the metadata page. If the database was opened non-transactionally the lock would get left behind. [#18257]

15. Fixed a bug that could lead to btree structure corruption if the DB->compact method ran out of locks [#18361]

16. Fixed a bug that would generate an error if a non-BDB file was used to create a database and the DB_TRUNCATE flag was specified. [#18373]

17. Fixed a bug that might cause a trap reading unitialized memory when backing out a merge of a duplicate tree leaf page during DB->compact. [#18461]

Locking Subsystem Changes

1. Fixed a bug to ensure deadlock detection works even when there are blocked transactions, configured with and without timeouts. [#17555]

2. Fixed a bug to ensure a call to the DB->key_range method from outside a transaction does not lock pages. [#17930]

3. Fixed a bug that could cause a segmentation fault if the lock manager ran out of mutexes [#18428]

Logging Subsystem Changes

1. Limited the size of a log record generated by freeing pages from a database, so that it fits in the log file size. [#17313]

Memory Pool Subsystem Changes

1. Fixed a bug to ensure mulitple versions of a buffer are not created when MVCC is not set. [#17495]

2. Fixed a bug to detect if cache size is being set when the cache is not configured. [#17556]

3. Fixed a bug to ensure the error message "unable to allocate space from the buffer cache" generated when there is still some space available, can be cleared by running recovery. [#17630]

4. Fixed a race condition that causes an operation to return EPERM when the buffer cache is nearly filled with pages belonging to recently closed queue extents. [#17840]

5. Fixed a bug that could cause a page needed by a snapshot reader to be overwritten rather than copied when it was freed. [#17973]

6. Enabled set_mp_pagesize to be specified in the DB_CONFIG file. [#18015]

7. Fixed a bug to ensure single-version or obsolete buffers were selected over any intermediate version. [#18114]

Mutex Subsystem Changes

1. Fixed a bug on HP-UX when specifying --with-mutex=HP/msem_init during configure. It would generate the error "TAS: mutex not appropriately aligned" at runtime, when initializing the first mutex. [#17489]

2. Fixed a race condition which could cause unnecessary retrying of btree searches when several threads simulatenously attempted to get a shared latch. [#18078]

3. Exclusive Transactions have been implemented for the SQL API. See the documentation for details on the behavior of this feature. [#17822]

Tcl-specific API Changes

1. Fixed a bug in Tcl API to prevent a segmentation fault from occurring when the get_dbname method is called to get the db name and the db handle is opened without providing either the filename or dbname. [#18037]

C#-specific API Changes

1. Fixed a bug in C# to prevent a System.AccessViolationException from occurring on Windows7 when trying to open new database. [#18422]

2. Fixed a bug in the C# API to make DB_COMPACT consistent with __db_compact in teh C API. [#18246]

API Changes

1. Added the dbstl_thread_exit method to release thread specific resouces on thread exit. [#17595]

2. Fixed the parser to allow configuration API flags set in the DB_CONFIG file to accept an optional ON/OFF string. The DB_REP_CONF_NOAUTOINIT flag has been removed. It is replaced by DB_REP_CONF_AUTOINIT. However, replication's default behavior remains the same. [#17795]

Replication Changes

1. Fixed bug where a not-in-sync client could service a peer request. [#18279]

2. Fixed bug where page gaps, once filled, would not immediately request the next page gap it finds. This was already fixed for logs. [#18219]

3. Fixed a bug so that only one thread waits for the meta-page lock during internal initialization and broadcasts out the information rather than all threads waiting. Removed the former retry code. [#17871]

4. Added a feature by which the DB->open method now allows the DB_CREATE flag on a replication client. It is ignored, but this allows a replication application to make one call that can work on either master or client. It fixes a possible race that could develop in a Replication Manager application if a call to DB->open is made around the same time as a master/client role change. [#15167]

5. The DB_ENV->repmgr_site_list method now returns an indication on whether the site is a client-to-client peer. [#16113]

6. Fixed a bug that could occasionally lead to elections failing to complete. [#17105]

7. Fixed a bug that could cause DB_ENV->txn_stat to trap. [#17198]

8. Added a new JOIN_FAILURE event to notify Replication Manager applications which refuse auto-initialization. [#17319]

9. Fixed a bug where a failed master lease check at a client site causes an ASSERT when processing a master lease grant at the master site. [#17869]

10. Fixed a bug to ensure a second simultaneous call to the DB_ENV->rep_elect method does not incorrectly clear a bit flag. [#17875]

11. Fixed a bug in client-side autoremoval of log files. [#17899]

12. Removed the likelihood of dual data streams to enhance network traffic. [#17955]

13. Fixed a bug such that non-txn dup cursors are accounted for in the replication API lockout. [#18080]

14. Fixed a bug to ensure checking for other sync states for the rerequest thread. [#18126]

15. Fixed a bug to avoid getting stuck in an election forever. [#18151]

16. Fixed a bug where using client-to-client synchronization with Master Leases could have resulted in failure of a new master to get initial lease grants from sufficient number of clients, resulting in a master environment panic. [#18254]

17. Fixed a bug which had prevented Replication Manager socket operations from working on HP/UX systems. [#18382]

18. Fixed a bug where starting as a client in multiple threads after receiving dupmaster messages could have resulted in a failure to find a new log file, resulting in a panic. [#18388]

19. The default thread stack size is no longer overridden by default for Berkeley DB threads. [#18383]

Transaction Subsystem Changes

1. Fixed a bug that caused transactions to deadlock on the mutex in the sequence object. [#17731]

2. Fixed a bug to ensure that the failure checking mechanism reconstructs child transactions correctly when a process dies with active sub-transactions. [#18154]

3. Removed a memory leak during recovery related to a deleted database [#18273]

Utility Changes

1. Fixed compiler warnings in the db_sql_codegen utility. [#17503]

2. Enhanced the db_recover -v utility to display the message, "No log files found", if no logs are present. [#17504]

3. Modified the db_verify utility to verify all files instead of aborting on the first failure. [#17513]

4. Modified the db_verify utility to display a message after verification is completed. [#17545]

5. Fixed a bug in the db_sql_codegen utility where the primary key is stored in both key and data fields. Removed it from the data field. [#17925]

Example Changes

1. Fixed a bug that causes the ex_txn C# example to hang. [#17376]
2. Fixed Solaris alignment issues in Stl port test code. [#17459]
3. Added GCC 4.4 compatibility support for all examples. [#17584]
4. Added new command line arguments(-h and -d) to the env examples. [#17624]
5. Fixed configuration problems related to running java API tests. [#17625]
6. Updated the bench_001 example to include bulk testing examples. [#17766]
7. Added a new Stl example to demo advanced feature usage. The Stl test cases referred earlier are replaced by these new examples in the Stl reference document. [#18175]

Deprecated Features

1. The configuration options --disable-cryptography and --enable-cryptoraphy are being deprecated. [#18110]

Configuration, Documentation, Sample Apps, Portability and Build Changes

1. Remove build files for Windows Visual Studio 6.0. [#16848]
2. Added an API, DBENV->db_full_version, to return db full version.
3. Berkeley DB no longer supports Win9X, Windows Me (Millenium edition) and NT 4.0. The minimum supported windows platform is Win 2k.
4. Berkeley DB no longer supports Visual Studio 6.0. The earliest version supported is Visual Studio 2005.
5. Added "+u1" to CFLAGS for HP ANSI C Compiler on HP-UX(IA64) to fix the alignment issue found with the allocation functions DB->set-alloc and DB_ENV->set_alloc. [#17257]
6. Fixed a bug such that the thread local storage (TLS) definition modifier is correctly deduced from the m4 script on all platforms. [#17609][#17713]
7. Fixed a bug such that TLS key is not initialized on platforms which do not support thread local storage (TLS) keywords, such as MAC OSX, and where TLS is implemented using pthread API. [#18001]
8. Fixed a bug to ensure that when using Intel C++ compiler (icpc), the TLS code builds successfully. A stricter criteria is adopted to deduce the TLS keyword, and hence pthread API is more likely to be used to implement TLS. [#18038]
9. Adding new configuration option, --with-cryptography={yes|no|ipp}. Using --with-cryptography=yes, will give equivalent behavior to the old --enable-cryptography

option. Using --with-cryptography=no, will give equivalent behavior to the old --disable-cryptography option. Using --with-cryptograhy=ipp will enable Intel's Performance Primitive (IPP) encryption on linux. [#18110]

Known Bugs

1. The configure option --with-uniquename may cause macro redefinition warnings on platforms where BDB implements parts of the standard C library. These warnings (e.g., "'db_int_def.h", line 586: warning: macro redefined: strsep') may occur when functions in the "clib" directory are included during configuration. This cosmetic affect does not affect the correct operation of the library. [#17172]

2. A multithreaded application using a private environment and multi-version concurrency control could, on very rare occasions, generate an illegal pointer access error during the final steps of a clean environment shutdown. [#17507]

3. Although rare, it is possible for a partial log record header at the end of a transaction log to be erroneously accepted as if it were valid, causing the error "Illegal record type 0 in log" during recovery. [#17851]

4. It is possible to get the error "unable to allocate space from the buffer cache" when there are disk errors on the freezer files used by multi-version concurrency control . [#17902]

5. Java API does not support partitioning by keys and the C# API doesn't support partitioning. [#18350]

6. If a database is removed from an environment and it was still opened transactionally and recovery is run, then a future recovery that must process that part of the log may fail. [#18459]

7. Replication "bulk transfer" does not work if Berkeley DB is unable to determine, at environment open time, whether the Replication Manager will be used. To work around this problem, an application using the Replication Manager should call DB_ENV->repmgr_set_local_site() before opening the environment. An application using the replication Base API should call DB_ENV->rep_set_transport() before opening the environment. [#18476]

8. The BTree prefix comparison function behaves slightly differently in the C API vs the C# API. In the C# API it returns a signed int and in the C API it returns an unsigned int. This can be a problem if the application needs to save more than 2^31 bytes.

Chapter 10. Upgrading Berkeley DB 4.7 applications to Berkeley DB 4.8

Introduction

The following pages describe how to upgrade applications coded against the Berkeley DB 4.7 release interfaces to the Berkeley DB 4.8 release interfaces. This information does not describe how to upgrade Berkeley DB 1.85 release applications.

Registering DPL Secondary Keys

Entity subclasses that define secondary keys must now be registered prior to storing an instance of the class. This can be done in two ways:

- The `EntityModel.registerClass()` method may be called to register the subclass before opening the entity store.
- The `EntityStore.getSubclassIndex()` method may be called to implicitly register the subclass after opening the entity store.

Failure to register the entity subclass will result in an `IllegalArgumentException` the first time an attempt is made to store an instance of the subclass. An exception will not occur if instances of the subclass have previously been stored, which allows existing applications to run unmodified in most cases.

This behavioral change was made to increase reliability. In several cases, registering an entity subclass has been necessary as a workaround. The requirement to register the subclass will ensure that such errors do not occur in deployed applications.

Minor Change in Behavior of DB_MPOOLFILE->get

DB 4.8 introduces some performance enhancements, based on the use of shared/exclusive latches instead of locks in some areas of the internal buffer management code. This change will affect how the `DB_MPOOL` interface handles dirty buffers.

Because of these changes, `DB_MPOOLFILE->get` will now acquire an exclusive latch on the buffer if the `DB_MPOOL_DIRTY` or `DB_MPOOL_EDIT` flags are specified. This could lead to an application deadlock if the application tries to fetch the buffer again, without an intervening `DB_MPOOLFILE->put` call.

If your application uses the `DB_MPOOL` interface, and especially the `DB_MPOOL_DIRTY` and `DB_MPOOL_EDIT` flags, you should review your code to ensure that this behavior change does not cause your application to deadlock.

Dropped Support for fcntl System Calls

Berkeley DB no longer supports mutex implementations based on the `fcntl` system call. If you have been configuring Berkeley DB to use this type of mutex, you need to either switch to a different mutex type or contact the Berkeley DB team for support.

Upgrade Requirements

The log file format changed in the Berkeley DB 4.8 release.

No database formats changed in the Berkeley DB 4.8 release.

The Berkeley DB 4.8 release does not support live replication upgrade from the 4.2 or 4.3 releases, only from the 4.4 and later releases.

For further information on upgrading Berkeley DB installations, see the Berkeley DB Upgrade Guide.

Berkeley DB 4.8.28 Change Log

Changes between 4.8.26 and 4.8.28:

1. Limit the size of a log record generated by freeing pages from a database so it fits in the log file size. [#17313]
2. Fix a bug that could cause a file to be removed if it was both the source and target of two renames within a transaction. [#18069]
3. Modified how we go about selecting a usable buffer in the cache. Place more emphasis on single version and obsolete buffers. [#18114]

Known bugs in 4.8

1. Sharing logs across mixed-endian systems does not work.[#18032]

Changes between 4.8.24 and 4.8.26:

1. Fixed a bug where the truncate log record could be too large when freeing too many pages during a compact. [#17313]
2. Fixed a bug where the deadlock detector might not run properly. [#17555]
3. Fixed three bugs related to properly detecting thread local storage for DbStl. [#17609] [#18001] [#18038]
4. Fixed a bug that prevented some of our example code from running correctly in a Windows environment. [#17627]
5. Fixed a bug where a "unable to allocate space from buffer cache" error was improperly generated. [#17630]
6. Fixed a bug where DB->exists() did not accept the DB_AUTO_COMMIT flag. [#17687]
7. Fixed a bug where DB_TXN_SNAPSHOT was not getting ignored when DB_MULTIVERSION not set. [#17706]
8. Fixed a bug that prevented callback based partitioning through the Java API. [#17735]

9. Fixed a replication bug where log files were not automatically removed from the client side. [#17899]

10. Fixed a bug where code generated from db_sql stored the key in both the data and key DBTs. [#17925]

11. Fixed a bug that prevented a sequence from closing properly after the EntityStore closed. [#17951]

12. Fixed a bug where gets fail if the DB_GET_BOTH_FLAG is specified in a hash, sorted duplicates database.[#17997]

Changes between 4.8.21 and 4.8.24:

1. Fixed a bug in the C# API where applications in a 64-bit environment could hang. [#17461]

2. Fixed a bug in MVCC where an exclusive latch was not removed when we couldn't obtain a buffer. [#17479]

3. Fixed a bug where a lock wasn't removed on a non-transactional locker. [#17509]

4. Fixed a bug which could trigger an assertion when performing a B-tree page split and running out of log space or with MVCC enabled. [#17531]

5. Fixed a bug in the repquote example that could cause the application to crash. [#17547]

6. Fixed a couple of bugs when using the GCC 4.4 compiler to build the examples and the dbstl API. [#17504] [#17476]

7. Fixed an incorrect representation of log system configuration info. [#17532]

Changes between 4.7 and 4.8.21:

Database or Log File On-Disk Format Changes:

1. The log file format changed in 4.8.

New Features:

1. Improved scalability and throughput when using BTree databases especially when running with multiple threads that equal or exceed the number of available CPUs.

2. Berkeley DB has added support for C#. In addition to the new C# api, C# specific tests and sample applications were also added. [#16137]

3. Berkeley DB has added an STL API, which is compatible with and very similar to C++ Standard Template Library (STL). Tests and sample applications and documentation were also added. [#16217]

4. Berkeley DB has added database partitioning. BTree or Hash databases may now be partitioned across multiple directories. Partitioned databases can be used to increase

concurrency and to improve performance by spreading access across disk subsystems. [#15862]

5. Berkeley DB now supports bulk insertion and deletion of data. Similar to the bulk get interface, the bulk put and bulk delete allow the developer to populate a buffer of key-value pairs and then pass it to the BDB library with a single API call.

6. Berkeley DB now supports compression when using BTree.

7. Berkeley DB introduces a new utility named db_sql which replaces db_codegen. Similar to db_codegen, db_sql accepts an input file with DDL statements and generates a Berkeley DB application using the C API that creates and performs CRUD operations on the defined tables. The developer can then use that code as a basis for further application development.

8. The Replication Manager now supports shared access to the Master database environment from multiple processes. In earlier versions, multiple process support on the Master required use of the Base Replication API. [#15982]

9. Foreign Key Support has been added to Berkeley DB.

10. Several enhancements were made to DB_REGISTER & DB_ENV->failchk().

11. Berkeley now supports 100% in-memory replication.

12. Berkeley DB now has the ability to compare two cursors for equality. [#16811]

Database Environment Changes:

1. Fixed a bug that could cause an allocation error while trying to allocate thread tracking information for the DB_ENV->failcheck system. [#16300]

2. Fixed a bug that could cause a trap if an environment open failed and failchk thread tracking was enabled.[#16770]

Concurrent Data Store Changes:

None.

General Access Method Changes:

1. Fixed a bug where doing an insert with secondary indices and the NOOVERWRITE flag could corrupt the secondary index. [#15912]

2. Fixed a possible file handle leak that occurred while aborting the create of a database whose metadata page was not initialized. [#16359]

3. Fixed a bug so that we now realloc the filename buffer only if we need it to grow. [#16385] [#16219]

4. Fixed a race freeing a transaction object when using MVCC. [#16381]

5. Added missing get methods for the DB and DB_ENV classes where there already was a corresponding set method. [#16505]

6. Fixed a bug to now ensure that DB_STAT_SUBSYSTEM is distinct from other stat flags. [#16798]

7. Fixed a bug related to updating multiple secondary keys (using DB_MULTIPLE). [#16885]

8. Fixed a bug so that verify (db->verify, db_verify) will now report when it cannot read a page rather than just saying the database is bad. [#16916]

9. Fixed a bug that could cause memory corruption if a transaction allocating a page aborted while DB->compact was running on that database. [#16862]

10. Fixed a bug where logging was occurring during remove of an in-memory database when the DB_TXN_NOT_DURABLE flag was set. [#16571]

11. Fixed a bug to remove a race condition during database/file create. [#17020]

12. Fixed a bug where a call to DB->verify and specifying DB_SALVAGE could leak memory when the call returned. [#17161]

13. Fixed a bug to avoid accessing freed memory during puts on primaries with custom comparators. [#17189]

14. Fixed a bug that could cause old versions of pages to be written over new versions if an existing database is opened with the DB_TRUNCATE flag. [#17191]

Btree Access Method Changes:

1. Fixed a bug which could cause DB->compact to fail with DB_NOTFOUND or DB_PAGE_NOTFOUND if the height of the tree was reduced by another thread while compact was active. The bug could also cause a page split to trigger splitting of internal nodes which did not need to be split. [#16192]

2. Fixed a bug that caused Db->compact to loop if run on an empty RECNO database when there were pages in the free list. [#16778]

3. Added a new flag, DB_OVERWRITE_DUP, to DB->put and DBC->put. This flag is equivalent to DB_KEYLAST in almost all cases: the exception is that with sorted duplicates, if a matching key/data pair exists, we overwrite it rather than returning DB_KEYEXIST. [#16803]

Hash Access Method Changes:

1. Fixed a bug to now force a group allocation that rolls forward to reinit all the pages. Otherwise a previous aborted allocation may change the header. [#15414]

2. Fixed a bug to now return the expected buffer size on a DB_BUFFER_SMALL condition. [#16881]

Queue Access Method Changes:

1. Fixed a bug that would cause the LSN reset functionality to not process queue extents. [#16213]

2. Fixed a bug that prevented a partial put on a queue database with secondaries configured. [#16460]

3. Fixed a bug to now prevent an unpinned page to be returned if a delete from a HASH database deadlocked. [#16371]

4. Fixed a bug that could cause a queue extent to be recreated if an application deleted a record that was already deleted in that extent. [#17004]

5. Added the DB_CONSUME flag to DB->del and DBC->del to force adjustment of the head of the queue. [#17004]

Recno Access Method Changes:

1. Fixed a bug which could cause DB->compact of a RECNO database to loop if the number of pages on the free list was reduced by another thread while compact was active. [#16199]

2. Fixed a bug that occurs when deleting from a Recno database and using DB_READ_UNCOMMITTED where we could try to downgrade a lock twice. [#16347]

3. Fixed a bug to now disallow passing DB_DUP and DB_RECNUM together to __db_set_flags. [#16585]

C-specific API Changes:

1. Add get functions for each set functions of DB and DB_ENV structures which didn't have one.[#16505]

C++-specific API Changes:

1. The get and set_lk_partitions methods are now available.

2. Add get functions for each set functions of Db and DbEnv classes which didn't have one. [#16505]

3. Fixed a memory leak when using nested transactions.[#16956]

Java-specific API Changes:

1. Fixed a bug where the replication finer-grained verbose flags were not available in the Java API. [#15419]

2. Fixed a bug in the BTree prefix compression API when called from the Java API. DBTs were not properly initialized. [#16417]

3. Fixed a bug so that LogCursor will work correctly from the Java API. [#16827]

4. Fixed a bug so that position(), limit() and capacity() of ByteBuffers are obeyed by DatabaseEntry objects. [#16982]

Direct Persistence Layer (DPL), Bindings and Collections API:

1. The StoredMap class now implements the standard java.util.concurrent.ConcurrentMap interface. [#15382]

2. Report a meaningful IllegalArgumentException when @Persistent is incorrectly declared on an enum class. Before, the confusing message Persistent class has non-persistent superclass: java.lang.Enum was reported. [#15623]

3. Report a meaningful IllegalArgumentException when @Persistent is incorrectly declared on an interface. Before, a NullPointerException was reported. [#15841]

4. Several validation checks have been added or corrected having to do with entity subclasses, which are @Persistent classes that extend an @Entity class. [#16077]

5. Optimized marshaling for large numbers of embedded objects improving performance. [#16198]

6. The StoredMap class now implements the Java 1.5 ConcurrentMap interface. [#16218]

7. Fix a DPL bug that caused exceptions when using a class Converter for an instance containing non-simple fields. [#16233]

8. Add EntityCursor.setCacheMode and getCacheMode. See the com.sleepycat.je.CacheMode class for more information. [#16239]

9. Fix a bug that prevents evolution of @SecondaryKey information in an entity subclass (a class that extends an @Entity class). [#16253]

10. Report a meaningful IllegalArgumentException when @Persistent or @Entity is incorrectly used on an inner class (a non-static nested class). Before, the confusing message No default constructor was reported. [#16279]

11. Improved the reliability of Entity subclasses that define secondary keys by requiring that they be registered prior to storing an instance of the class. [#16399]

12. Fix a bug that under certain circumstances causes "IllegalArgumentException: Not a key class" when calling EntityStore.getSubclassIndex, EntityStore.getPrimaryConfig, EntityStore.getSecondaryConfig, or PrimaryIndex.put, and a composite key class is used. [#16407]

13. Fixed a bug so that one can now compile DPL in the Java API on Windows. [#16570]

14. The com.sleepycat.collections.TransactionRunner.handleException method has been added to allow overriding the default transaction retry policy. See the javadoc for this method for more information. [#16574]

15. Fix a bug that causes an assertion to fire or a NullPointerException (when assertions are disabled) from the EntityStore constructor. The problem occurs only when the previously

created EntityStore contains an entity with a secondary key definition in which the key name has been overridden and is different than the field name. [#16819]

16. Key cursors have been optimized to significantly reduce I/O when the READ_UNCOMMITTED isolation mode is used. See EntityIndex.keys for more information. [#16859]

17. Report a meaningful IllegalArgumentException when NULLIFY is used with a @SecondaryKey and the field is a primitive type. Before, the confusing message Key field object may not be null was reported. [#17011]

18. Enum fields may now be used as DPL keys, including primary keys, secondary keys, and fields of composite key classes. Comparators are supported for composite key classes containing enum fields. [#17140]

19. Fix a bug that prevented the use of custom key comparisons (composite key classes that implement Comparable) for secondary keys defined as ONE_TO_MANY or MANY_TO_MANY. [#17207]

20. The db.jar file now contains a Premain class which enables bytecode enhancement using the JVM instrumentation commands. The built-in proxy classes are also now enhanced in the db.jar file, which enables off-line bytecode enhancement. For more information on DPL bytecode enhancement and how to use both instrumentation and off-line enhancement, please see the com.sleepycat.persist.model.ClassEnhancer javadoc. [#17233]

Tcl-specific API Changes:

1. The mutex API is now available when using Tcl. [#16342]

RPC-specific Client/Server Changes:

- RPC support has been removed from Berkeley DB. [#16785]

Replication Changes:

1. Improved testing of initial conditions for rep and repmgr APIs and added heartbeat timeouts to rep_get_timeout.[#14977]

2. Added DB_REP_CONF_INMEM replication configuration flag to store replication information exclusively in-memory without creating any files on-disk. [#15257]

3. Added repmgr support for multi-process shared env [#15982]

4. Fixed a bug where opening a cursor from a database handle failed to check whether the database handle was still fresh. If the database handle had been invalidated by a replication client synchronizing with a new master, it could point to invalid information. [#15990]

5. Fixed a bug so that if LOG_REQ gets an archived LSN, replication sends VERIFY_FAIL. [#16004]

6. Added timestamp and process/thread id to replication verbose messages. [#16098]

7. Fixed a bug where, in very rare circumstances, two repmgr sites could connect to each other at the exact same time, the connection attempts "collide" and fail, and the same collision repeats in time synchronization indefinitely. [#16114]

8. Fixed a bug where a missing database file (FILE_FAIL error condition) can interrupt a client synchronization without restarting it. [#16130]

9. Fixed a bug by adding REP_F_INREPSTART flag to prevent racing threads in rep_start. [#16247]

10. Fixed a bug to not return HOLDELECTION if we are already in the middle of an election. Updated the egen so the election thread will notice. [#16270]

11. Fixed a bug in buffer space computation, which could have led to memory corruption in rare circumstances, when using bulk transfer. [#16357]

12. Fixed a bug that prevented replication clients from opening a sequence. The sequence is opened for read operations only. [#16406]

13. Fixed a bug by removing an assertion about priority in elections. It is not correct because it could have changed by then. Remove unused recover_gen field. [#16412]

14. Fixed a bug to now ignore a message from client if it is an LSN not recognized in a LOG_REQ. [#16444]

15. Fixed a bug so that on POSIX systems, repmgr no longer restores default SIGPIPE action upon env close, if it was necessary to change it during start-up. This allows remaining repmgr environments within the same process, if any, to continue operating after one of them is closed. [#16454]

16. After a replication client restarts with recovery, any named in-memory databases are now re-materialized from the rest of the replication group upon synchronization with the master. [#16495]

17. Fixed a bug by adding missing rep_get_config flags. [#16527]

18. Instead of sleeping if the bulk buffer is in transmission, return so that we can send as a singleton. [#16537]

19. Fixed a bug by changing __env_refresh to not hit assert on -private -rep env with an in-memory database. [#16546]

20. Fixed a bug in the Windows implementation of repmgr where a large number of commit threads concurrently awaiting acknowledgments could result in memory corruption, and leaking Win32 Event Objects. [#16548]

21. Fixed a bug by changing repmgr to count a dropped connection when noticing a lacking heartbeat; fixed hearbeat test to check for election, rather than connection drop count, and more reasonable time limit; fixed test to poll until desired result, rather than always sleeping max possible time. [#16550]

22. Fixed "master changes" stat to count when local site becomes master too. [#16562]

23. Fixed a bug where a c2c client would send UPDATE_REQ to another client [#16592]

24. Removed code to proactively expire leases when we don't get acks. Leases maintain their own LSNs to know. [#16494]

25. Fixed a bug where a client may not sync pages during internal init. [#16671]

26. Fixed a bug where a client that received and skipped a log record from the master during an election, then won the election, could then try to request a copy of the skipped log record. The result was an attempt to send a request to the local site, which is invalid: this could confuse a replication Base API application, or cause the Replication Manager to crash. [#16700]

27. Fixed a bug which could have caused data loss or corruption (at the client only) if a replication client rolled back existing transactions in order to synchronize with a new master, and then crashed/recovered before a subsequent checkpoint operation had been replicated from the master. [#16732]

28. Fixed a bug so that replication now retries on DB_LOCK_NOTGRANTED. [#16741]

29. Fixed a potential deadlock in rep_verify_fail. [#16779]

30. Fixed a bug so that an application will no longer segv if nsites given was smaller than number of sites that actually exists. [#16825]

XA Resource Manager Changes:

1. The XA Resource Manager has been removed from Berkeley DB. [#6459]

Locking Subsystem Changes:

1. Fixed a bug to prevent unlocking a mutex twice if we ran out of transactional locks. [#16285]

2. Fixed a bug to prevent a segmentation trap in __lock_open if there were an error during the opening of an environment. [#16307]

3. Fixed a bug to now avoid a deadlock if user defined locks are used only one lock partition is defined.[#16415]

4. Fixed concurrency problems in __dd_build, __dd_abort by adding LOCK_SYSTEM_LOCK() calls to __dd_build and __dd_abort. [16489]

5. Fixed a bug that could cause a panic if a transaction which updated a database that was supporting READ_UNCOMMITED readers aborted and it hit a race with a thread running the deadlock detector. [#16490]

6. Fixed a race condition in deadlock detection that could overwrite heap. [#16541]

7. Fixed a bug so that DB_STAT_CLEAR now restores the value of st_partitions. [#16701]

Logging Subsystem Changes:

1. Fixed a bug so that the header checksum is only ignored when the log is from a previous version [#16281]
2. Fixed a bug by removing a possible race condition with logc_get(DB_FIRST) and log archiving. [#16387]
3. Fixed a bug that could cause a recovery failure of a create of a database that was aborted. [#16824]
4. An in-memory database creation has an intermediate phase where we have a semi-open DBP. If we crash in that state, then recovery was failing because it tried to use a partically open database handle. This fix checks for that case, and avoids trying to undo page writes for databases in that interim step. [#17203]

Memory Pool Subsystem Changes:

1. Fixed a bug that occurred after all open handles on a file are closed. Needed to clear the TXN_NOT_DURABLE flag (if set) and mark the file as DURABLE_UNKNOWN in the memory pool. [#16091]
2. Fixed a possible race condition between dirtying and freeing a buffer that could result in a panic or corruption. [#16530]
3. Fixed a memory leak where allocated space for temporary file names are not released. [#16956]

Mutex Subsystem Changes:

1. Fixed a bug when using mutexes for SMP MIPS/Linux systems. [#15914]
2. POSIX mutexes are now the default on Solaris. [#16066]
3. Fixed a bug in mutex allocation with multiple cache regions. [#16178]
4. Fixed MIPS/Linux mutexes in 4.7. [#16209]
5. Fixed a bug that would cause a mutex to be unlocked a second time if we ran out of space while tracking pinned pages. [#16228]
6. Fixed a bug Sparc/GCC when using test-and-set mutexes. They are now aligned on an 8-byte boundary. [#16243]
7. Fixed a bug to now prevent a thread calling DB_ENV->failcheck to hang on a mutex held by a dead thread. [#16446]
8. Fixed a bug so that __db_pthread_mutex_unlock() now handles the failchk case of finding a busy mutex which was owned by a now-dead process. [#16557]
9. Removed support for the mutex implementation based on the "fcntl" system call. Anyone configuring Berkeley DB to use this type of mutex in an earlier release will need to either switch to a different mutex type or contact Oracle for support. [#17470]

Test Suite Changes

1. Fixed a bug when using failchk(), where a mutex was not released. [#15982]
2. Added a set of basic repmgr tests to run_std and run_all. [#16092]
3. Added control wrapper for db_reptest to test suite. [#16161]
4. Fixed a bug to now skip tests if db_reptest is not configured. [#16161]
5. Changed name of run_db_in_mem to run_inmem_db, and run_inmem to run_inmem_log and made the arg orders consistent. [#16358]
6. Fixed a bug to now clean up stray handles when rep_verify doesn't work. [#16390]
7. Fixed a bug to avoid db_reptest passing the wrong flag to repmgr_start when there is already a master. [#16475]
8. Added new tests for abbreviated internal init. Fixed test not to expect in-memory database to survive recovery. [#16495]
9. Fix a bug, to add page size for txn014 if the default page size is too small. Move files instead of renaming directory for env015 on QNX. [#16627]
10. Added new rep088 test for log truncation integrity. [#16732]
11. Fixed a bug by adding a checkpoint in rep061 to make sure we have messages to process. Otherwise we could hang with client stuck in internal init, and no incoming messages to trigger rerequest. [#16781]

Transaction Subsystem Changes:

1. Fixed a bug to no longer generate an error if DB_ENV->set_flags (DB_TXN_NOSYNC) was called after the environment was opened. [#16492]
2. Fixed a bug to remove a potential hang condition in replication os_yield loops when DB_REGISTER used with replication by adding PANIC_CHECKS. [#16502]
3. Fix a bug to now release mutex obtained before special condition returns in __db_cursor_int and __txn_record_fname. [#16665]
4. Fixed a leak in the transaction region when a snapshot update transaction accesses more than 4 databases. [#16734]
5. Enabled setting of set_thread_count via the DB_CONFIG file. [#16878]
6. Fixed a mutex leak in some corner cases. [#16665]

Utility Changes:

1. The db_stat utility with the -RA flags will now print a list of known remote replication flags when using repmgr. [#15484]

2. Restructured DB salvage to walk known leaf pages prior to looping over all db pages. [#16219]

3. Fixed a problem with upgrades to 4.7 on big endian machines. [#16411]

4. Fixed a bug so that now db_load consistently returns >1 on failure. [#16765]

5. The db_dump utility now accepts a "-m" flag to dump information from a named in-memory database. [#16896]

6. Fixed a bug that would cause db_hotbackup to fail if a database file was removed while it was running. [#17234]

Configuration, Documentation, Sample Application, Portability and Build Changes:

1. Fixed a bug to now use the correct Perl include path. [#16058]

2. Updated the version of the Microsoft runtime libraries shipped. [#16058]

3. Upgraded the Visual Studio build files to be based on Visual Studio 8 (2005+). The build is now simplified. Users can still upgrade the Visual Studio 6.0 project files, if they want to use Visual Studio .NET (7.1) [#16108]

4. Expanded the ex_rep example with checkpoint and log archive threads, deadlock detection, new options for acknowledgment policy and bulk transfer, and use of additional replication features and events. [#16109]

5. Fixed a bug so that optimizations on AIX are re-enabled, avoiding incorrect code generation. [#16141]

6. Removed a few compiler warnings and three type redefinitons when using vxworks and the GNU compiler. [#16341]

7. Fixed a bug on Sparc v9 so that MUTEX_MEMBAR() now uses membar_enter() to get a #storeload barrier rather than just stbar's #storestor. [#16468]

8. Berkeley DB no longer supports Win9X and Windows Me (Millenium edition).

9. Fixed lock_get and lock_vec examples from the Java (and C#) API. Updated the Java lock example. [#16506]

10. Fixed a bug to correctly handle the TPC-B history record on 64-bit systems. [#16709]

11. Add STL API to Linux build. Can be enabled via the --enable-stl flag. [#16786]

12. Add STL API to Windows build, by building the db_stl project in the solution. There are also stl's test and examples projects in this solution. [#16786]

13. Add support to build dll projects for WinCE, in order to enable users to build DB into a dll in addition to a static library.[#16625]

14. Fixed a weakness where several malloc/realloc return values are not checked before use. [#16664]

15. Enabled DB->compact for WinCE.[#15897]

16. HP-UX 10 is no longer supported.

Chapter 11. Test Suite

Running the test suite

Once you have started tclsh and have loaded the test.tcl source file (see Running the test suite under UNIX (page 53) and Running the test suite under Windows (page 27) for more information), you are ready to run the test suite. At the tclsh prompt, to run the standard test suite, enter the following:

```
% run_std
```

A more exhaustive version of the test suite runs all the tests several more times, testing encryption, replication, and different page sizes. After you have a clean run for run_std, you may choose to run this lengthier set of tests. At the tclsh prompt, enter:

```
% run_all
```

Running the standard tests can take from several hours to a few days to complete, depending on your hardware, and running all the tests will take at least twice as long. For this reason, the output from these commands are redirected to a file in the current directory named `ALL.OUT`. Periodically, a line will be written to the standard output, indicating what test is being run. When the test suite has finished, a final message will be written indicating the test suite has completed successfully or that it has failed. If the run failed, you should review the `ALL.OUT` file to determine which tests failed. Errors will appear in that file as output lines, beginning with the string "FAIL".

Tests are run in the directory `TESTDIR`, by default. However, the test files are often large, and you should use a filesystem with at least several hundred megabytes of free space. To use a different directory for the test directory, edit the file include.tcl in your build directory, and change the following line to a more appropriate value for your system:

```
set testdir ./TESTDIR
```

For example, you might change it to the following:

```
set testdir /var/tmp/db.test
```

Alternatively, you can create a symbolic link named TESTDIR in your build directory to an appropriate location for running the tests. Regardless of where you run the tests, the TESTDIR directory should be on a local filesystem. Using a remote filesystem (for example, an NFS mounted filesystem) will almost certainly cause spurious test failures.

Running SQL Test Suite on Unix

Once the test suite is built (see Building SQL Test Suite on Unix (page 53) for more information), run the entire test suite by executing the following command in the `../build_unix/sql` directory:

```
sh ../../sql/adapter/bdb-test.sh
```

This runs a set of tests and lists the errors each test encountered, if any. A detailed list of the test results is written to `test.log`.

To run an individual test, such as insert.test, execute the following command in the `../build_unix/sql` directory:

```
./textfixture ../../sql/sqlite/test/insert.test
```

Running SQL Test Suite on Windows

After the test suite is built (see Building the software needed by the SQL tests (page 28) for more information) and before running the entire test suite, go to `../sql/adapter/bdb-test.sh` and edit the line:

```
echo $t: `alarm $TIMEOUT ./testfixture.exe
$tpath 2>&1 | tee -a test.log | grep "errors out of"
|| echo "failed"`
```

to

```
echo $t: `alarm $TIMEOUT Win32/Debug/testfixture.exe
$tpath 2>&1 | tee -a test.log | grep "errors out of"
|| echo "failed"`
```

Running the test suite requires an Unix emulator, such as Cygwin. In a Cygwin window go to the `../build_windows` directory and execute the command:

```
sh ../sql/adapter/bdb-test.sh
```

This runs a set of tests and lists errors that each test encountered, if any. A detailed list of the test results is written to `test.log`.

To run an individual test, such as insert.test, execute the following command in the `../build_windows` directory:

```
Win32/Debug/testfixture.exe ../sql/sqlite/test/insert.test
```

Test suite FAQ

1. **The test suite has been running for over a day. What's wrong?**

   The test suite can take anywhere from some number of hours to several days to run, depending on your hardware configuration. As long as the run is making forward progress and new lines are being written to the `ALL.OUT` files, everything is probably fine.