SWI-Prolog BerkeleyDB interface

Jan Wielemaker
VU university Amsterdam
The Netherlands
E-mail: J.Wielemaker@vu.nl

July 4, 2016

Abstract

This package realised external storage of Prolog terms based on the *Berkeley DB* library from Sleepycat Software. The DB library implements modular support for the bottom layers of a database. The database itself maps unconstrained keys onto values.

The SWI-Prolog interface for DB allows for fast storage of arbitrary Prolog terms in the database.

Contents

1 Introduction

The native Prolog database is not very well suited for either *very* large data-sets or dynamically changing large data-sets that need to be communicated between Prolog instances or need to be safely guarded against system failure. These cases ask for an external database that can be attached quickly and provides protection against system failure.

The Berkeley DB package is an open source library realising the bottom-layers of a database. It is a modular system, which in it's simplest deals with resource management on a mapped file and in its most complex form deals with network transparency, transaction management, locking, recovery, life-backup, etc.

The DB library maps keys to values. Optionally multiple values can be associated with a key. Both key and value are arbitrary-length binary objects.

This package stores arbitrary Prolog terms into the database, serializing them using `PL_record_external()`. This provides an interface similar to the recorded-database (`recorda/3`), which supports terms with internal sharing, cycles and attributes. In addition, it can store restricted data types such as atoms, strings and integers using standard representations which allows for sharing the database with other languages.

1.1 About this manual

This manual is by no means complete. The Berkeley DB documentation should be consulted directly to resolve details on security, resource usage, formats, configuration options etc. This interface passed default values for most DB API calls. Supported options hint to the corresponding DB API calls, which should be consulted for details.

2 library(bdb): Berkeley DB interface

This package realises a binding to *Berkeley DB*, originally by Sleepycat Software, now managed by Oracle. The DB library implements modular support for the bottom layers of a database. In can be configured for single-threaded access to a file, multi-threaded access with transactions, remote access as well as database replication.

Berkeley DB is an *embedded* database. This implies the library provides access to a file containing one or more database tables. The Berkeley DB database tables are always *binary*, mapping a *key* to a *value*. The SWI-Prolog interface to Berkeley DB allows for fast storage of arbitrary Prolog terms including cycles and constraints in the database.

Accessing a database consists of four steps:

1. Initialise the default DB environment using `bdb_init/1` or create an explicit DB environment using `bdb_init/2`. This step is optional, providing simple non-transactional file access when omitted.

2. Open a database using `bdb_open/4`, returning a handle to the database.

3. Accessing the data using `bdb_put/3`, `bdb_get/3`, etc.

4. Closing a database using `bdb_close/1`. When omitted, all open databases are closed on program halt (see `at_halt/1`).

Errors reported by the underlying database are mapped to an exception of the form `error(bdb(Code,Message,Object), _)`, where *Code* is an atom for well known errors and an integer for less known ones. *Message* is the return from the `db_strerror()` function and *Object* is the most related Prolog object, typically a database or database environment handle. If *Code* is an atom, it is the lowercase version of the associated C macro after string the `DB_` prefix. Currently the following atom-typed codes are defined: `lock_deadlock`, `runrecovery`, `notfound`, `keyempty`, `keyexist`, `lock_notgranted` and `secondary_bad`.

bdb_init(*+Options*) *[det]*

bdb_init(*-Environment, +Options*) *[det]*

Initialise a DB *environment*. The predicate `bdb_init/1` initialises the *default* environment, while `bdb_init/2` creates an explicit environment that can be passed to `bdb_open/4` using the `environment(+Environment)` option. If `bdb_init/1` is called, it must be called before the first call to `bdb_open/4` that uses the default environment. If `bdb_init/1` is not called, the default environment can only handle plain files and does not support multiple threads, locking, crash recovery, etc.

Initializing a BDB environment always requires the `home(+Dir)` option. If the environment contains no databases, the argument `create(true)` must be supplied as well.

The currently supported options are listed below. The name of the boolean options are derived from the DB flags by dropping the =DB_= prefix and using lowercase, e.g. `DB_INIT_LOCK` becomes `init_lock`. For details, please refer to the DB manual.

create(*+Bool*)
: If `true`, create any underlying file as required. By default, no new files are created. This option should be set for prograns that create new databases.

failchk(*+Bool*)

home(*+Home*)
: Specify the DB home directory, the directory holding the database files. The directory must exist prior to calling these predicates.

init_lock(*+Bool*)
: Enable locking (`DB_INIT_LOCK`). Implied if transactions are used.

init_log(*+Bool*)
: Enable logging the DB modifications (`DB_INIT_LOG`). Logging enables recovery of databases in case of system failure. Normally it is used in combination with transactions.

init_mpool(*+Bool*)
: Initialize memory pool. Impicit if `mp_size(+Size)` or `mp_mmapsize(+Size)` is specified.

init_rep(*+Bool*)
: Init database replication. The rest of the replication logic is not yet supported.

init_txn(*+Bool*)
: Init transactions. Implies `init_log(true)`.

lockdown(*+Bool*)

mp_size(*+Integer*)

mp_mmapsize(*+Integer*)
Control memory pool handling (`DB_INIT_MPOOL`). The `mp_size` option sets the memory-pool used for caching, while the `mp_mmapsize` controls the maximum size of a DB file mapped entirely into memory.

private(*+Bool*)

recover(*+Bool*)
Perform recovery before opening the database.

recover_fatal(*+Bool*)
Perform fatal recovery before opening the database.

register(*+Bool*)

server(*+Host, [+ServerOptions]*)
Initialise the DB package for accessing a remote database. *Host* specifies the name of the machine running `berkeley_db_svc`. Optionally additional options may be specified:

server_timeout(*+Seconds*)
Specify the timeout time the server uses to determine that the client has gone. This implies the server will terminate the connection to this client if this client does not issue any requests for the indicated time.

client_timeout(*+Seconds*)
Specify the time the client waits for the server to handle a request.

system_mem(*+Bool*)

transactions(*+Bool*)
Enable transactions, providing atomicy of changes and security. Implies logging and locking. See `bdb_transaction/1`.

thread(*+Bool*)
Make the environment accessible from multiple threads.

thread_count(*+Integer*)
Declare an approximate number of threads in the database environment. See `DB_ENV->set_thread_count()`.

use_environ(*+Bool*)

use_environ_root(*+Bool*)

config(*+ListOfConfig*)
Specify a list of configuration options, each option is of the form Name(Value). Currently unused.

bdb_close_environment(*+Environment*) *[det]*

Close a database environment that was explicitly created using `bdb_init/2`.

bdb_current_environment(*-Environment*) *[nondet]*

True when *Environment* is a currently known environment.

bdb_environment_property(*?Environment, ?Property*) *[nondet]*

True when *Property* is a property of *Environment*. Defined properties are all boolean options defined with `bdb_init/2` and the following options:

home(*-Path*)

Path is the absolute path name for the directory used as database environment.

open(*-Boolean*)

True if the environment is open.

bdb_open(*+File, +Mode, -DB, +Options*) *[det]*

Open *File* holding a database. *Mode* is one of `read`, providing read-only access or `update`, providing read/write access. *Options* is a list of options. Supported options are below. The boolean options are passed as flags to `DB->open()`. The option name is derived from the flag name by stripping the `DB_` prefix and converting to lower case. Consult the Berkeley *DB* documentation for details.

auto_commit(*+Boolean*)

Open the database in a transaction. Ensures no database is created in case of failure.

create(*+Boolean*)

Create a new database of the database does not exist.

dup(*+Boolean*)

Do/do not allow for duplicate values on the same key. Default is not to allow for duplicates.

excl(*+Boolean*)

Combined with `create(true)`, fail if the database already exists.

multiversion(*+Boolean*)

Open the database with support for multiversion concurrency control. The flag is passed, but no further support is provided yet.

nommap(*+Boolean*)

Do not map this database into process memory.

rdonly(*+Boolean*)

Open the database for reading only.

read_uncommitted(*+Boolean*)

Read operations on the database may request the return of modified but not yet committed data. This flag must be specified on all *DB* handles used to perform dirty reads or database updates, otherwise requests for dirty reads may not be honored and the read may block.

thread(*+Boolean*)
Enable access to the database handle from multiple threads. This is default if the corresponding flag is specified for the environment.

truncate(*+Boolean*)
When specified, truncate the underlying file, i.e., start with an empty database.

database(*+Name*)
If *File* contains multiple databases, address the named database in the file. A *DB* file can only consist of multiple databases if the `bdb_open/4` call that created it specified this argument. Each database in the file has its own characteristics.

environment(*+Environment*)
Specify a database environment created using `bdb_init/2`.

key(*+Type*)

value(*+Type*)
Specify the type of the key or value. Allowed values are:

term
Key/Value is a Prolog term (default). This type allows for representing arbitrary Prolog data in both keys and value. The representation is space-efficient, but Prolog specific. See PL_record_external() in the SWI-Prolog Reference Manual for details on the representation. The other representations are more neutral. This implies they are more stable and sharing the *DB* with other languages is feasible.

atom
Key/Value is an atom. The text is represented as a UTF-8 string and its length.

c_blob
Key/Value is a blob (sequence of bytes). On output, a Prolog string is used. The input is either a Prolog string or an atom holding only characters in the range [0..255].

c_string
Key/Value is an atom. The text is represented as a C 0-terminated UTF-8 string.

c_long
Key/Value is an integer. The value is represented as a native C long in machine byte-order.

Arguments

DB	is unified with a *blob* of type `db`. Database handles are subject to atom garbage collection.

Errors `permission_error(access, bdb_environment, Env)` if an environment is not thread-enabled and accessed from multiple threads.

bdb_close(*+DB*) *[det]*
Close BerkeleyDB database indicated by *DB*. *DB* becomes invalid after this operation.

An attempt to access a closed database is detected reliably and results in a permission_error exception.

bdb_put(*+DB, +Key, +Value*) *[det]*
Add a new key-value pair to the database. If the database does not allow for duplicates the possible previous associated with *Key* is replaced by *Value*.

bdb_del(*+DB, ?Key, ?Value*) *[nondet]*
Delete the first matching key-value pair from the database. If the database allows for duplicates, this predicate is non-deterministic, otherwise it is *semidet*. The enumeration performed by this predicate is the same as for `bdb_get/3`. See also `bdb_delall/3`.

bdb_delall(*+DB, +Key, ?Value*) *[det]*
Delete all matching key-value pairs from the database. With unbound *Value* the key and all values are removed efficiently.

bdb_get(*+DB, ?Key, -Value*) *[nondet]*
Query the database. If the database allows for duplicates this predicate is non-deterministic, otherwise it is *semidet*. Note that if *Key* is a term this matches stored keys that are *variants* of *Key*, **not** unification. See `=@=/2`. Thus, after `bdb_put(DB, f(X), 42)`, we get the following query results:

- `bdb_get(DB, f(Y), V)` binds *Value* to `42`, while *Y* is left unbound.
- `bdb_get(DB, f(a), V)` *fails*.
- `bdb_enum(DB, f(a), V)` succeeds, but does not perform any indexing, i.e., it enumerates all key-value pairs and performs the unification.

bdb_enum(*+DB, -Key, -Value*)
Enumerate the whole database, unifying the key-value pairs to *Key* and *Value*. Though this predicate can be used with an instantiated *Key* to enumerate only the keys unifying with *Key*, no indexing is used by `bdb_enum/3`.

bdb_getall(*+DB, +Key, -Values*) *[semidet]*
Get all values associated with *Key*. Fails if the key does not exist (as `bagof/3`).

bdb_current(*?DB*) *[nondet]*
True when *DB* is a handle to a currently open database.

bdb_closeall *[det]*
Close all currently open databases and environments. This is called automatically after loading this library on process terminatation using `at_halt/1`.

bdb_transaction(*:Goal*) *[semidet]*
bdb_transaction(*+Environment, :Goal*) *[semidet]*
Start a transaction, execute *Goal* and terminate the transaction. Only if *Goal* succeeds, the transaction is commited. If *Goal* fails or raises an exception, the transaction is aborted and `bdb_transaction/1` either fails or rethrows the exception. Of special interest is the exception

```
error(package(db, deadlock), _)
```

This exception indicates a deadlock was raised by one of the DB predicates. Deadlocks may arise if multiple processes or threads access the same keys in a different order. The DB infra-structure causes one of the processes involved in the deadlock to abort its transaction. This process may choose to restart the transaction.

For example, a DB application may define `{Goal}` to realise transactions and restart these automatically is a deadlock is raised:

```
{Goal} :-
    catch(bdb_transaction(Goal), E, true),
    (   var(E)
    ->  true
    ;   E = error(package(db, deadlock), _)
    ->  {Goal}
    ;   throw(E)
    ).
```

Arguments

Environment	defines the environment to which the transaction applies. If omitted, the default environment is used. See `bdb_init/1` and `bdb_init/2`.

bdb_version(-*Version:integer*) *[det]*

True when *Version* identifies the database version. *Version* is an integer defined as:

```
DB_VERSION_MAJOR*10000 +
DB_VERSION_MINOR*100   +
DB_VERSION_PATCH
```

3 Copyright and license

The SWI-Prolog interface code is licensed under the same conditions as SWI-Prolog itself.

The following is the license that applies to this copy of the Berkeley DB software. For a license to use the Berkeley DB software under conditions other than those described here, or to purchase support for this software, please contact Oracle at berkeleydb-info_us@oracle.com.

```
/*
 * Copyright (c) 1990, 2010 Oracle and/or its affiliates.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
```

```
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Redistributions in any form must be accompanied by information on
 *    how to obtain complete source code for the DB software and any
 *    accompanying software that uses the DB software.  The source code
 *    must either be included in the distribution or be available for no
 *    more than the cost of distribution plus a nominal fee, and must be
 *    freely redistributable under reasonable conditions.  For an
 *    executable file, complete source code means the source code for all
 *    modules it contains.  It does not include source code for modules or
 *    files that typically accompany the major components of the operating
 *    system on which the executable file runs.
 *
 * THIS SOFTWARE IS PROVIDED BY ORACLE ``AS IS'' AND ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR
 * NON-INFRINGEMENT, ARE DISCLAIMED.  IN NO EVENT SHALL ORACLE BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
 * BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
 * IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
/*
 * Copyright (c) 1990, 1993, 1994, 1995
 *      The Regents of the University of California.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
```

```
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */
/*
 * Copyright (c) 1995, 1996
 *      The President and Fellows of Harvard University.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY HARVARD AND ITS CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL HARVARD OR ITS CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
/***
 * ASM: a very small and fast Java bytecode manipulation framework
 * Copyright (c) 2000-2005 INRIA, France Telecom
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
```

```
*    notice, this list of conditions and the following disclaimer in the
*    documentation and/or other materials provided with the distribution.
* 3. Neither the name of the copyright holders nor the names of its
*    contributors may be used to endorse or promote products derived from
*    this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*/
```