SWI-Prolog External Database

Jan Wielemaker
SWI,
University of Amsterdam
The Netherlands
E-mail: `jan@swi.psy.uva.nl`

June 26, 2000

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. Both key and value are *binary blobs*.

The SWI-Prolog interface for DB allows for fast storage of general 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 by SleepyCat software is a GPL'ed 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, using PL_record_external() introduced in SWI-Prolog 3.3.7, in the database. It provides an interface similar to the recorded-database (`recorda/3`). In the future we plan to link this interface transparently to a predicate.

2 The DB interface

2.1 Opening a database

db_open(*+File, +Mode, -DB, +Options***)**

Open a 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. Currently supported options are:

duplicates(*bool***)**

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

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 `db_open/4` call that created it specified this argument. Each database in the file has its own characteristics.

key(*Type***)**

Specify the type of the key. Allowed values are:

term

Key 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 is an atom. The text is represented using the character data and its length.

c_string

Key is an atom. The text is represented as a C 0-terminated string.

c_long
Key is an integer. The value is represented as a native C long in the machines byte-order.

value(*Type***)**
Specify the type of the value. See `key` for details.

2.2 Accessing a database

db_put(*+DB, +Key, +Value***)**
Add a new key-value pair to the database. If the database allows for duplicates this will always succeed, unless a system error occurs.

db_del(*+DB, +Key***)**
Delete all key-value pairs with the named *Key* from the database. Note that this is equivalent to `db_delall`(*DB, Key, _*).

db_del(*+DB, ?Key, ?Value***)**
Delete the first matching key-value pair from the database. If the database allows for duplicates, this predicate is non-deterministic. The enumeration performed by this predicate is the same as for `db_get/3`.

db_delall(*+DB, +Key, ?Value***)**
Delete all matching key-value pairs from the database. With unbound key this calls `db_del/2`.

db_get(*+DB, ?Key, -Value***)**
Query the database. If the database allows for duplicates this predicate is non-deterministic. If *Key* is unbound, all keys are enumerated. This mode is especially useful to enumerating the entire content of a database.

db_getall(*+DB, +Key, -Value***)**
Get all values associated with *Key*. Fails if the key does not exist (as `bagof/3`).

2.3 Transactions

Using the DB transaction protocol, security against system failure, atomicy of multiple changes and accessing a database from multiple writers is provided.

Accessing a DB under transactions from Prolog is very simple. First of all, the option `transactions`(*true*) needs to be provided to `db_init/1` to initialise the transaction subsystem. Next, the predicate `db_transaction/1` may be used to execute multiple updates inside a transaction.

db_transaction(*:Goal***)**
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 `db_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 `{}/1` to realise transactions and restart these automatically is a deadlock is raised:

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

2.4 Notes on signals and other interrupts

The Berkeley DB routines are not signal-safe. Without precaution, this implies it is not possible to interrupt Prolog programs using the DB routines in a safe manner. To improve convinience, interrupt signals are blocked during the execution of the DB calls. As `db_transaction/1` handles aborts gracefully, PrologDB applications can be interrupted and aborted safely.

Signals other than `SIGINT` caught during the execution of one of the DB interaction predicates may leave the DB in an inconsistent state. Fatal signals thrown by other Prolog or foreign language facilities are handled gracefully.

2.5 Initialisation

db_init(*+Options*)

Initialise the DB package. This must be done before the first call to `db_open/4` and at maximum once. If `db_open/4` is called without calling `db_init/1`, default initialisation is used, which is generally suitable for handling small databases without support for advanced features.

Options is a list of options. The currently supported are listed below. For details, please refer to the DB manual.

home(*Home*)
: Specify the DB home directory, the directory holding the database files.

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

mp_size(*+Integer*)
: Size of the memory-pool used for caching.

mp_mmapsize(*+Integer*)
: Maximum size of a DB file mapped entirely into memory.

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

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

transactions(*+Bool*)
: Enable transactions, providing atomicy of changes and security. Implies `logging` and `locking`. See section 2.3.

3 Installation

3.1 DB version

This package was developed for DB version 3.1. The interface of DB 3.x is fundamentally different from previous releases and db4pl relies on functionality provided by DB 3.x. Unfortunately many distributions of DB are still based on DB 2.x. Please make sure to install DB 3.1 or later before building db4pl.

3.2 Unix systems

Installation on Unix system uses the commonly found *configure*, *make* and *make install* sequence. SWI-Prolog should be installed before building this package. If SWI-Prolog is not installed as `pl`, the environment variable `PL` must be set to the name of the SWI-Prolog executable. Installation is now accomplished using:

```
% ./configure
% make
% make install
```

This installs the foreign libraries in `$PLBASE/lib/$PLARCH` and the Prolog library files in `$PLBASE/library`, where `$PLBASE` refers to the SWI-Prolog 'home-directory'.

Configure recognises the following options in addition to the default GNU configure options.

--enable-mt
: Enable thread-support for the multi-threaded version of SWI-Prolog. Currently not supported.

--with-db=*DIR*
: Point to the installation-directory of DB 3.x for finding include files and the DB libraries. For example:

```
./configure --with-db=/usr/local/BerkeleyDB.3.1
```