## NAME

DBD::File - Base class for writing file based DBI drivers

## SYNOPSIS

This module is a base class for writing other DBDs. It is not intended to function as a DBD itself (though it is possible). If you want to access flat files, use DBD::AnyData, or DBD::CSV (both of which are subclasses of DBD::File).

## DESCRIPTION

The DBD::File module is not a true DBI driver, but an abstract base class for deriving concrete DBI drivers from it. The implication is, that these drivers work with plain files, for example CSV files or INI files. The module is based on the SQL::Statement module, a simple SQL engine.

See DBI for details on DBI, SQL::Statement for details on SQL::Statement and DBD::CSV, DBD::DBM or DBD::AnyData for example drivers.

### Metadata

The following attributes are handled by DBI itself and not by DBD::File, thus they all work as expected:

```
Active
ActiveKids
CachedKids
CompatMode (Not used)
InactiveDestroy
AutoInactiveDestroy
Kids
PrintError
RaiseError
Warn (Not used)
```

*The following DBI attributes are handled by DBD::File:*

AutoCommit

Always on.

ChopBlanks

Works.

NUM_OF_FIELDS

Valid after `$sth->execute`.

NUM_OF_PARAMS

Valid after `$sth->prepare`.

NAME

Valid after `$sth->execute`; undef for Non-Select statements.

NULLABLE

Not really working, always returns an array ref of ones, except the affected table has been created in this session. Valid after `$sth->execute`; undef for non-select statements.

*Unsupported DBI attributes and methods*

bind_param_inout

CursorName

LongReadLen

LongTruncOk

*DBD::File specific attributes*

In addition to the DBI attributes, you can use the following dbh attributes:

f_dir

This attribute is used for setting the directory where the files are opened and it defaults to the current directory (.). Usually you set it on the dbh but it may be overridden per table (see f_meta).

When the value for `f_dir` is a relative path, it is converted into the appropriate absolute path name (based on the current working directory) when the dbh attribute is set.

```
f_dir => "/data/foo/csv",
```

See "KNOWN BUGS AND LIMITATIONS".

f_dir_search

This optional attribute can be set to pass a list of folders to also find existing tables. It will **not** be used to create new files.

```
f_dir_search => [ "/data/bar/csv", "/dump/blargh/data" ],
```

f_ext

This attribute is used for setting the file extension. The format is:

```
extension{/flag}
```

where the /flag is optional and the extension is case-insensitive. `f_ext` allows you to specify an extension which:

```
f_ext => ".csv/r",
```

- makes DBD::File prefer *table.extension* over *table*.
- makes the table name the filename minus the extension.

```
DBI:CSV:f_dir=data;f_ext=.csv
```

In the above example and when `f_dir` contains both *table.csv* and *table*, DBD::File will open *table.csv* and the table will be named "table". If *table.csv* does not exist but *table* does that file is opened and the table is also called "table".

If `f_ext` is not specified and *table.csv* exists it will be opened and the table will be called "table.csv" which is probably not what you want.

NOTE: even though extensions are case-insensitive, table names are not.

```
DBI:CSV:f_dir=data;f_ext=.csv/r
```

The `r` flag means the file extension is required and any filename that does not match the extension is ignored.

Usually you set it on the dbh but it may be overridden per table (see f_meta).

f_schema

This will set the schema name and defaults to the owner of the directory in which the table file resides. You can set `f_schema` to `undef`.

```
my $dbh = DBI->connect ("dbi:CSV:", "", "", {
f_schema => undef,
f_dir => "data",
f_ext => ".csv/r",
}) or die $DBI::errstr;
```

By setting the schema you affect the results from the tables call:

```
my @tables = $dbh->tables ();

# no f_schema
```

```
"merijn".foo
"merijn".bar

# f_schema => "dbi"
"dbi".foo
"dbi".bar

# f_schema => undef
foo
bar
```

Defining `f_schema` to the empty string is equal to setting it to `undef` so the DSN can be `"dbi:CSV:f_schema=;f_dir=."`.

f_lock

The `f_lock` attribute is used to set the locking mode on the opened table files. Note that not all platforms support locking. By default, tables are opened with a shared lock for reading, and with an exclusive lock for writing. The supported modes are:

```
0: No locking at all.

1: Shared locks will be used.

2: Exclusive locks will be used.
```

But see KNOWN BUGS below.

f_lockfile

If you wish to use a lockfile extension other than `.lck`, simply specify the `f_lockfile` attribute:

```
$dbh = DBI->connect ("dbi:DBM:f_lockfile=.foo");
$dbh->{f_lockfile} = ".foo";
$dbh->{dbm_tables}{qux}{f_lockfile} = ".foo";
```

If you wish to disable locking, set the `f_lockfile` to `0`.

```
$dbh = DBI->connect ("dbi:DBM:f_lockfile=0");
$dbh->{f_lockfile} = 0;
$dbh->{dbm_tables}{qux}{f_lockfile} = 0;
```

f_encoding

With this attribute, you can set the encoding in which the file is opened. This is implemented using `binmode $fh, ":encoding(<f_encoding>)"`.

f_meta

Private data area aliasing "sql_meta" in DBI::DBD::SqlEngine which contains information about the tables this module handles. Table meta data might not be available until the table has been accessed for the first time e.g., by issuing a select on it however it is possible to pre-initialize attributes for each table you use.

DBD::File recognizes the (public) attributes `f_ext`, `f_dir`, `f_file`, `f_encoding`, `f_lock`, `f_lockfile`, `f_schema`, in addition to the attributes "sql_meta" in DBI::DBD::SqlEngine already supports. Be very careful when modifying attributes you do not know, the consequence might be a destroyed or corrupted table.

`f_file` is an attribute applicable to table meta data only and you will not find a corresponding attribute in the dbh. Whilst it may be reasonable to have several tables with the same column names, it is not for the same file name. If you need access to the same file using different table names, use `SQL::Statement` as the SQL engine and the `AS` keyword:

```
SELECT * FROM tbl AS t1, tbl AS t2 WHERE t1.id = t2.id
```

`f_file` can be an absolute path name or a relative path name but if it is relative, it is interpreted as being relative to the `f_dir` attribute of the table meta data. When `f_file` is set DBD::File will use `f_file` as specified and will not attempt to work out an alternative for `f_file` using the `table name`and `f_ext` attribute.

While `f_meta` is a private and readonly attribute (which means, you cannot modify it's values), derived drivers might provide restricted write access through another attribute. Well known accessors are `csv_tables` for DBD::CSV `ad_tables` for DBD::AnyData and `dbm_tables` for DBD::DBM

*New opportunities for attributes from DBI::DBD::SqlEngine*

sql_table_source

`$dbh->{sql_table_source}` can be set to *DBD::File::TableSource::FileSystem* (and is the default setting of DBD::File). This provides usual behaviour of previous DBD::File releases on

```
@ary = DBI->data_sources ($driver);
@ary = DBI->data_sources ($driver, \%attr);

@ary = $dbh->data_sources ();
@ary = $dbh->data_sources (\%attr);

@names = $dbh->tables ($catalog, $schema, $table, $type);

$sth = $dbh->table_info ($catalog, $schema, $table, $type);
$sth = $dbh->table_info ($catalog, $schema, $table, $type, \%attr);

$dbh->func ("list_tables");
```

sql_data_source

`$dbh->{sql_data_source}` can be set to either *DBD::File::DataSource::File* which is default and provides the well known behavior of DBD::File releases prior to 0.41, or *DBD::File::DataSource::Stream* which reuses already opened file-handle for operations.

*Internally private attributes to deal with SQL backends*

Do not modify any of these private attributes unless you understand the implications of doing so. The behavior of DBD::File and derived DBDs might be unpredictable when one or more of those attributes are modified.

sql_nano_version

Contains the version of loaded DBI::SQL::Nano.

sql_statement_version

Contains the version of loaded SQL::Statement.

sql_handler

Contains either the text 'SQL::Statement' or 'DBI::SQL::Nano'.

sql_ram_tables

Contains optionally temporary tables.

sql_flags

Contains optional flags to instantiate the SQL::Parser parsing engine when SQL::Statement is used as SQL engine. See SQL::Parser for valid flags.

### Driver private methods

*Default DBI methods*

data_sources

The `data_sources` method returns a list of subdirectories of the current directory in the form “dbi:CSV:f_dir=$dirname”.

If you want to read the subdirectories of another directory, use

```
my ($drh) = DBI->install_driver ("CSV");
my (@list) = $drh->data_sources (f_dir => "/usr/local/csv_data");
```

*Additional methods*

The following methods are only available via their documented name when DBD::File is used directly. Because this is only reasonable for testing purposes, the real names must be used instead. Those names can be computed by replacing the `f_` in the method name with the driver prefix.

f_versions

Signature:

```
sub f_versions (;$)
{
my ($table_name) = @_;
$table_name ||= ".";
...
}
```

Returns the versions of the driver, including the DBI version, the Perl version, DBI::PurePerl version (if DBI::PurePerl is active) and the version of the SQL engine in use.

```
my $dbh = DBI->connect ("dbi:File:");
my $f_versions = $dbh->func ("f_versions");
print "$f_versions\n";
_ _END_ _
# DBD::File 0.41 using IO::File (1.16)
# DBI::DBD::SqlEngine 0.05 using SQL::Statement 1.406
# DBI 1.623
# OS darwin (12.2.1)
# Perl 5.017006 (darwin-thread-multi-ld-2level)
```

Called in list context, f_versions will return an array containing each line as single entry.

Some drivers might use the optional (table name) argument and modify version information related to the table (e.g. DBD::DBM provides storage backend information for the requested table, when it has a table name).

## KNOWN BUGS AND LIMITATIONS

- This module uses flock () internally but flock is not available on all platforms. On MacOS and Windows 95 there is no locking at all (perhaps not so important on MacOS and Windows 95, as there is only a single user).
- The module stores details about the handled tables in a private area of the driver handle (`$drh`). This data area is not shared between different driver instances, so several `DBI->connect ()`calls will cause differen t table instances and private data areas.

  This data area is filled for the first time when a table is accessed, either via an SQL statement or via `table_info` and is not destroyed until the table is dropped or the driver handle is released. Manual destruction is possible via f_clear_meta.

  The following attributes are preserved in the data area and will evaluated instead of driver globals:

f_ext
f_dir
f_dir_search
f_lock
f_lockfile
f_encoding
f_schema
col_names
sql_identifier_case

The following attributes are preserved in the data area only and cannot be set globally.

f_file

The following attributes are preserved in the data area only and are computed when initializing the data area:

f_fqfn
f_fqbn
f_fqln
table_name

For DBD::CSV tables this means, once opened "foo.csv" as table named "foo", another table named "foo" accessing the file "foo.txt" cannot be opened. Accessing "foo" will always access the file "foo.csv" in memorized `f_dir`, locking `f_lockfile` via memorized `f_lock`.

You can use f_clear_meta or the `f_file` attribute for a specific table to work around this.

- When used with SQL::Statement and temporary tables e.g.,

  ```
  CREATE TEMP TABLE ...
  ```

  the table data processing bypasses DBD::File::Table. No file system calls will be made and there are no clashes with existing (file based) tables with the same name. Temporary tables are chosen over file tables, but they will not covered by `table_info`.

## AUTHOR

This module is currently maintained by

H.Merijn Brand < h.m.brand at xs4all.nl > and Jens Rehsack < rehsack at googlemail.com >

The original author is Jochen Wiedmann.

## COPYRIGHT AND LICENSE

## SEE ALSO

DBI, DBD::DBM DBD::CSV Text::CSV, Text::CSV_XS, SQL::Statement, and DBI::SQL::Nano