

NAME

DBD::File::HowTo - Guide to create DBD::File based driver

SYNOPSIS

```
perldoc DBD::File::HowTo
perldoc DBI
perldoc DBI::DBD
perldoc DBD::File::Developers
perldoc DBI::DBD::SqlEngine::Developers
perldoc DBI::DBD::SqlEngine
perldoc SQL::Eval
perldoc DBI::DBD::SqlEngine::HowTo
perldoc SQL::Statement::Embed
perldoc DBD::File
perldoc DBD::File::HowTo
perldoc DBD::File::Developers
```

DESCRIPTION

This document provides a step-by-step guide, how to create a new [DBD::File](#) based DBD. It expects that you carefully read the DBI documentation and that you're familiar with [DBI::DBD](#) and had read and understood [DBD::ExampleP](#).

This document addresses experienced developers who are really sure that they need to invest time when writing a new DBI Driver. Writing a DBI Driver is neither a weekend project nor an easy job for hobby coders after work. Expect one or two man-month of time for the first start.

Those who are still reading, should be able to sing the rules of "CREATING A NEW DRIVER" in [DBI::DBD](#)

Of course, [DBD::File](#) is a [DBI::DBD::SqlEngine](#) and you surely read [DBI::DBD::SqlEngine::HowTo](#) before continuing here.

CREATING DRIVER CLASSES

Do you have an entry in DBI's DBD registry? For this guide, a prefix of `foo_` is assumed.

Sample Skeleton

```
package DBD::Foo;

    use strict;
    use warnings;
    use vars qw(@ISA $VERSION);
    use base qw(DBD::File);

    use DBI ();

    $VERSION = "0.001";

    package DBD::Foo::dr;

    use vars qw(@ISA $imp_data_size);

    @ISA = qw(DBD::File::dr);
    $imp_data_size = 0;

    package DBD::Foo::db;

    use vars qw(@ISA $imp_data_size);

    @ISA = qw(DBD::File::db);
```

```

$imp_data_size = 0;

package DBD::Foo::st;

use vars qw(@ISA $imp_data_size);

@ISA = qw(DBD::File::st);
$imp_data_size = 0;

package DBD::Foo::Statement;

use vars qw(@ISA);

@ISA = qw(DBD::File::Statement);

package DBD::Foo::Table;

use vars qw(@ISA);

@ISA = qw(DBD::File::Table);

1;

```

Tiny, eh? And all you have now is a DBD named foo which will be able to deal with temporary tables, as long as you use SQL::Statement. In [DBI::SQL::Nano](#) environments, this DBD can do nothing.

Start over

Based on [DBI::DBD::SqlEngine::HowTo](#), we're now having a driver which could do basic things. Of course, it should now derive from [DBD::File](#) instead of [DBI::DBD::SqlEngine](#), shouldn't it?

[DBD::File](#) extends [DBI::DBD::SqlEngine](#) to deal with any kind of files. In principle, the only extensions required are to the table class:

```

package DBD::Foo::Table;

sub bootstrap_table_meta
{
my ( $self, $dbh, $meta, $table ) = @_;

# initialize all $meta attributes which might be relevant for
# file2table

return $self->SUPER::bootstrap_table_meta($dbh, $meta, $table);
}

sub init_table_meta
{
my ( $self, $dbh, $meta, $table ) = @_;

# called after $meta contains the results from file2table
# initialize all missing $meta attributes

$self->SUPER::init_table_meta( $dbh, $meta, $table );
}

```

In case [DBD::File::Table::open_file](#) doesn't open the files as the driver needs that, override

```

it!

sub open_file
{
my ( $self, $meta, $attrs, $flags ) = @_;
# ensure that $meta->{f_dontopen} is set
$self->SUPER::open_file( $meta, $attrs, $flags );
# now do what ever needs to be done
}

```

Combined with the methods implemented using the `SQL::Statement::Embed` guide, the table is full working and you could try a start over.

User comfort

`DBD::File` since 0.39 consolidates all persistent meta data of a table into a single structure stored in `$dbh->{f_meta}`. With `DBD::File` version 0.41 and `DBI::DBD::SqlEngine` version 0.05, this consolidation moves to `DBI::DBD::SqlEngine`. It's still the `$dbh->{$drv_prefix . "_meta"}` attribute which cares, so what you learned at this place before, is still valid.

```

sub init_valid_attributes
{
my $dbh = $_[0];

$dbh->SUPER::init_valid_attributes ();

$dbh->{foo_valid_attrs} = { ... };
$dbh->{foo_readonly_attrs} = { ... };

$dbh->{foo_meta} = "foo_tables";

return $dbh;
}

```

See updates at “User comfort” in `DBI::DBD::SqlEngine::HowTo`.

Testing

Now you should have your own `DBD::File` based driver. Was easy, wasn't it? But does it work well? Prove it by writing tests and remember to use `dbd_edit_mm_attribs` from `DBI::DBD` to ensure testing even rare cases.

AUTHOR

This guide is written by Jens Rehsack. `DBD::File` is written by Jochen Wiedmann and Jeff Zucker.

The module `DBD::File` is currently maintained by

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

COPYRIGHT AND LICENSE

Copyright (C) 2010 by H.Merijn Brand & Jens Rehsack

All rights reserved.

You may freely distribute and/or modify this module under the terms of either the GNU General Public License (GPL) or the Artistic License, as specified in the Perl README file.