Database Abstraction Layer

Extension Key: **dbal**

Copyright 2004-2006, Karsten Dambekalns, <karsten@typo3.org>
Copyright 2004, Kasper Skårhøj, <kasperYYYY@typo3.com>

Table of Contents

Introduction

What does it do?

It provides a layer for support of other core databases (DBMS) in TYPO3 than the default database support based on MySQL. Further, each layer can be selected on a per-table basis, thus offering storage of content from TYPO3 in multiple databases and multiple formats.

The support is possible through the well known PHP database API ADOdb. The extension also supports any user-defined layer you can make yourself thus offering unlimited possibilities for connectivity.

Technical details

This extension works only with >= TYPO3 3.8.0 since that version contains a complete implementation of a general database interaction wrapper class, t3lib_DB, which this extension extends (by the “ux_” method, thus containing the class “ux_t3lib_DB”).

Basically TYPO3 3.8.0 allows extensions to supply DBAL because all calls to the database is done through an instance of the class t3lib_DB. That class contains basic wrapper functions for MySQL and some special, recommended functions as well. It has been available since TYPO3 3.6.0RC2, but has changed internally over time.

The wrapper class means that TYPO3 without DBAL extensions installed will work as always, and with virtually no overhead in database connectivity while *offering* DBAL for those who need it through extensions like this one!

Working databases

The following databases have been tested and are know to work in general. This means, not every aspect has been thoroughly checked, but TYPO3 has been installed from scratch successfully, BE logins are possible, and basic websites can be created (template, pages, content elements):

- MySQL 4.0.x and 5.0.x
- PostgreSQL 7.x and 8.x
- Oracle 8, 9 and 10
- Firebird 1.5.2

Configuration

Introduction

Before the DBAL will do anything different for you than just connecting to the default database you will have to configure it. By default it connects using the “native” handler type - which means direct interaction with MySQL.

Since the DBAL offers to store information in multiple sources and not just a single database you might have to understand handlers first.

First, some definitions:

- **handler type** - which kind of interface is used for a data handler. The options are “native”, “adodb” or “userdefined”.
    - native - Connects directly to MySQL with hardcoded PHP functions
    - adodb - Is an instance of ADOdb database API offering support for a long list of databases other than MySQL. The DBAL extension has been developed with a focus on ADOdb until now, so it should work.
    - userdefined - Is an instance of a userdefined class which must contain certain functions to supply results from the “database” - offers support for just any kind of data source you can program an interface to yourself!
- **handlerKey** - a string which uniquely identifies a data handler. Each handler represents an instance of a handler type (see above). The handlerKey can be any alphanumeric string. The handler key “_DEFAULT” is the default handler for all tables unless otherwise configured.
- **tablename** - the database table name seen from the TYPO3 side in the system (might differ from the *real* database name if mapping is enabled!)

$TYPO3_CONF_VARS['EXTCONF']['dbal']

The DBAL is configurable through $TYPO3_CONF_VARS['EXTCONF']['dbal'] *entered in “ext_localconf.php” / “localconf.php”.* This table is an overview of the main keys in this array:

Key	Datatype:	Description:
handlerCfg[*handlerKey*]	->handlerCfg	Configuration of each data handler you want to use in the system. Each handler is identified with a string (handlerKey) which is used in the “table2handlerKeys” configuration (see below) to pair tablenames with handlers. There is *always* a default handler needed which has the handlerKey “_DEFAULT”. By default this handler is configured with the classic username/password/host and database settings from localconf.php in TYPO3. If you want to use ADOdb or just need to store a table in another database you can configure a handler here and map the tables you need to that handler (with “table2handlerKeys”, see below).
table2handlerKeys[*tablename*]	handlerKey	Using other handlers than the “_DEFAULT” handler key is possible on a per-table basis and simply done by entering the table name as key in this array and letting the value be the handlerKey you want to use for this table! **Notice:** If tables are joined *both tables* must use the same handlerKey. If they do not TYPO3 will exit with a fatal error! You can use the debug options to track all table joins and assess which tables can safely be handled together.

Key	Datatype:	Description:
mapping[*tablename*]	->mapping	Configuration of mapping of table and fieldnames. For instance you can configure that TYPO3 should use a physical table in the database named “typo3_pages” instead of “pages”. Or you can map fieldname in a similar fashion. The point is that TYPO3 always sees a table or field names as TYPO3 requires internally but in reality the table- or field name could be something different in the physical database source. There is a performance loss by configuring such mapping of course: Result rows are preprocessed before being returned and all SQL queries are parsed, transformed and re-compiled again before execution.
debugOptions	->debug	Options for various debugging in the DBAL.

->handlerCfg

Configuration of a data handler

Key	Datatype:	Description:
type	handler type (string)	The type of the handler. The type is a fixed keyword between these: • native • adodb • userdefined (See description of each in the introduction above) The “native” handler is used by default (and is MySQL-only!) The handler type will determine what options are available for “config”
config	array	Array containing configuration for the handler. See below for options. Notice that the options are supported depending on handler type. For this, see information in italic and square brackets.
config[username]	string	Username for connection **Notice:** For the “_DEFAULT” handler this is overridden by $typo_db_username from localconf.php *[Only native / adodb]*
config[password]	string	Password for connection **Notice:** For the “_DEFAULT” handler this is overridden by $typo_db_password from localconf.php *[Only native / adodb]*
config[host]	string	Host for the database server **Notice:** For the “_DEFAULT” handler this is overridden by $typo_db_host from localconf.php *[Only native / adodb]*
config[port]	integer	Port for the database server *[Only native / adodb]*
config[database]	string	The database name **Notice:** For the “_DEFAULT” handler this is overridden by $typo_db from localconf.php *[Only native / adodb]*
config[driver]	string	Which driver, (eg. “mysql”, “oci8” etc.). Depending on API (see ADOdb documentation for details) *[Only adodb]*
config[sequenceStart]	integer	The number which is used as initial value for sequences when they are generated. *[Only adodb]*
config[classFile]	string	Class file for user defined DB handler class. Eg. “EXT:dbal/handlers/class.tx_dbal_handler_xmldb.php” Must be relative path to PATH_site. The “EXT:” prefix can be used for locations inside of extensions. *[Only userdefined]*

Key	Datatype:	Description:
config[class]	string	Class name for the handler inside of config[classFile]. Eg. “tx_dbal_handler_xmldb” Please see examples/templates of userdefined handlers inside dbal/handlers/ directory. *[Only userdefined]*

Using ADOdb or PEAR::DB for the _DEFAULT handler

```
 1: $TYPO3_CONF_VARS['EXTCONF']['dbal']['handlerCfg'] = array (
 2:         '_DEFAULT' => array (
 3:             'type' => 'adodb',
 4:             'config' => array(
 5:                 'driver' => 'mysql',
 6:             )
 7:         )
 8:     );
```

If you need to use other databases, just change the value in line 5 to the name of the other database driver. See ADOdb manual for details.

Using another MySQL database for the “tt_guest” and “sys_note” tables

```
 1: $TYPO3_CONF_VARS['EXTCONF']['dbal']['handlerCfg'] = array (
 2:         '_DEFAULT' => array (
 3:             'type' => 'native',
 4:             'config' => array(
 5:                 'username' => '',        // Set by default (overridden)
 6:                 'password' => '',        // Set by default (overridden)
 7:                 'host' => '',            // Set by default (overridden)
 8:                 'database' => '',        // Set by default (overridden)
10:             )
11:         ),
12:         'alternativeMySQLdb' => array (
13:             'type' => 'native',
14:             'config' => array(
15:                 'username' => 'your_username',
16:                 'password' => 'your_password',
17:                 'host' => 'localhost',
18:                 'database' => 'alternative_database_name',
19:             )
20:         ),
21:     );
22:
23: $TYPO3_CONF_VARS['EXTCONF']['dbal']['table2handlerKeys'] = array (
24:         'tt_guest' => 'alternativeMySQLdb',
25:         'sys_note' => 'alternativeMySQLdb',
26:     );
```

In line 24 and 25 we configure the two tables to use the *handler key* “alternativeMySQLdb” instead of the “_DEFAULT” handler. In both cases the handlers will connect natively to MySQL - but two different databases at the “same time”.

Storing “tt_guest” and “sys_note” tables in Oracle

```
$TYPO3_CONF_VARS['EXTCONF']['dbal']['handlerCfg'] = array (
        '_DEFAULT' => array (
            'type' => 'native',
            'config' => array(
                'username' => '',        // Set by default (overridden)
                'password' => '',        // Set by default (overridden)
                'host' => '',            // Set by default (overridden)
                'database' => '',        // Set by default (overridden)
            )
        ),
        'oracleDB' => array (
            'type' => 'adodb',
            'config' => array(
                'username' => 'your_username',
                'password' => 'your_password',
                'host' => 'localhost',
                'database' => 'oracleDB',
                'driver' => 'oci8'
            )
        ),
    );
```

```
22:
23: $TYPO3_CONF_VARS['EXTCONF']['dbal']['table2handlerKeys'] = array (
24:         'tt_guest' => 'oracleDB',
25:         'sys_note' => 'oracleDB',
26:     );
```

This example is basically similar to the former, just that the key name was changed to “oracleDB” for convenience.

The real change is that

- line 12 configures ADOdb to be used and
- line 18 configures ADOdb to use the “oci8” driver instead of MySQL.

Storing “tt_guest” and “sys_note” tables in an XML file

```
$TYPO3_CONF_VARS['EXTCONF']['dbal']['handlerCfg'] = array (
        '_DEFAULT' => array (
            'type' => 'native',
            'config' => array(
                'username' => '',        // Set by default (overridden)
                'password' => '',        // Set by default (overridden)
                'host' => '',            // Set by default (overridden)
                'database' => '',        // Set by default (overridden)
            )
        ),
        'xmlDB' => array (
            'type' => 'userdefined',
            'config' => array(
                'classFile' => 'EXT:dbal/handlers/class.tx_dbal_handler_xmldb.php',
                'class' => 'tx_dbal_handler_xmldb',
                'tableFiles' => array(
                    'tt_guest' => 'fileadmin/tt_guest.xml',
                    'sys_note' => 'fileadmin/sys_note.xml',
                )
            )
        ),
    );

$TYPO3_CONF_VARS['EXTCONF']['dbal']['table2handlerKeys'] = array (
        'tt_guest' => 'xmlDB',
        'sys_note' => 'xmlDB',
    );
```

In this example the handler key “xmlDB” sets up a userdefined handler; basically a PHP class with certain functions for INSERT / UPDATE / DELETE and SELECT operations and data-to-disc I/O. In this case it is just an example using the class “tx_dbal_handler_xmldb” which is shipped with this extensions. Configuration might be different since that class (at time of writing) is not finished.

Anyways, the point is that this userdefined, PHP written handler will simulate an SQL server and allow to insert, select, update and delete records which is actually stored in some XML files and not real database tables!

This goes to show the possibilities, right... :-)

Notice on joins and tables separated into different databases

If you chose to configure that some tables like “sys_note” and “tt_guest” will go into other databases as the example shows above you will have to make sure *they are never joined with any tables from other databases*. If they are, you will face a fatal error from the DBAL; logically you cannot join tables across database systems!

->mapping

Contains mapping of tablename and fields in a table. Notice that entering any configuration for a table might affect performance since translation is needed before results are returned or queries executed.

Mapping is totally transparent for applications inside TYPO3 and mapping is independent of handler type - the translation goes on between these two spheres.

Mapping can work as a work-around for reserved field- or table names.

Key	Datatype:	Description:
mapTableName	string	Real, physical tablename for the table
mapFieldNames[fieldname]	string	Real, physical fieldname in the table.

Example

```
$TYPO3_CONF_VARS['EXTCONF']['dbal']['mapping'] = array (
        'sys_note' => array (
```

```
            'mapTableName' => 'SysNoteTable',
            'mapFieldNames' =>  array (
                'uid' => 'uid999',
                'pid' => 'pid999',
                'deleted' => 'deleted999',
                'tstamp' => 'tstamp999',
                'crdate' => 'crdate999',
                'cruser' => 'cruser999',
                'author' => 'author999',
                'email' => 'email999',
                'subject' => 'subject999',
                'message' => 'message999',
                'personal' => 'personal999',
                'category' => 'category999'
            )
        ),
        '_tt_content' => array (
            'mapTableName' => 'tt_content999',
            'mapFieldNames' => array (
                'bodytext' => 'bodytext999',
                'header' => 'header999',
                'image' => 'image999',
                'pid' => 'pid999',
                'sorting' => 'sorting999',
            )
        )
    );
```

In this example two classic TYPO3 tables have been mapped; the “sys_note” table (from the “sys_note” extension) and the “tt_content” table (Content Elements).

According to this mapping example the sys_note table in the database (or whatever data source) is actually named “SysNoteTable” and all fields are actually named differently; with “...999” after (this is just an example).

When you try to make a look up in the sys_note like “SELECT uid FROM sys_note WHERE uid=123” then this is transformed into “SELECT uid999 FROM SysNoteTable WHERE uid999=123” before executed. And the result row which will be array('uid999' => 123) will be transformed back to array('uid' => 123) before you receive it inside of TYPO3.

Notice: Mapping tables to two different databases on localhost will most likely only work if [SYS][no_pconnect] is set in $TYPO3_CONF_VARS. Otherwise PHP will, regardless of DBAL maintaining different links for the databases, use the wrong one.

->debug

Debugging options

Key	Datatype:	Description:
enabled	boolean	If set, TYPO3 will log every SQL execution in the tx_dbal_debuglog table. This option must be set for the other options below to work. You can view the log from the backend; There is a DBAL module in the Tools main module.
printErrors	boolean	If set, SQL errors will be debug()'ed to browser after any SQL execution.
EXPLAIN	boolean	Will log the result of a EXPLAIN SELECT... in case of select-queries. Can help you to benchmark the performance of you indices in the database.
parseQuery	boolean	Will parse all possible parts of the SQL queries, compile them again and match the results. If the parsed and recompiled queries did not match they will enter the log table and can subsequently be addressed. This will help you to spot “TYPO3 incompatible SQL” (as defined by the core parser of “t3lib_sqlengine”).
joinTables	boolean	Will log every SELECT query performed with a table join - necessary to make sure that all tables that may be joined in TYPO3 is also handled by the same handlerKey (which is required for obvious reasons!)

Example

This enables all debug options:

```
$TYPO3_CONF_VARS['EXTCONF']['dbal']['debugOptions']     = array(
        'enabled' => TRUE,        // Generally, enable debugging.
        'printErrors' => TRUE,    // Enable output of SQL errors after query executions.
        'EXPLAIN' => 1,           // EXPLAIN SELECT ...(Only on default handler)
        'parseQuery' => 1,        // Parsing queries, testing parsability (All queries)
        'joinTables' => 1
    );
```

Database-specific configuration hints

Depending on the database you use, the meaning of the host/username/password settings may slightly differ. This the case for Oracle, and maybe other RDBMS as well.

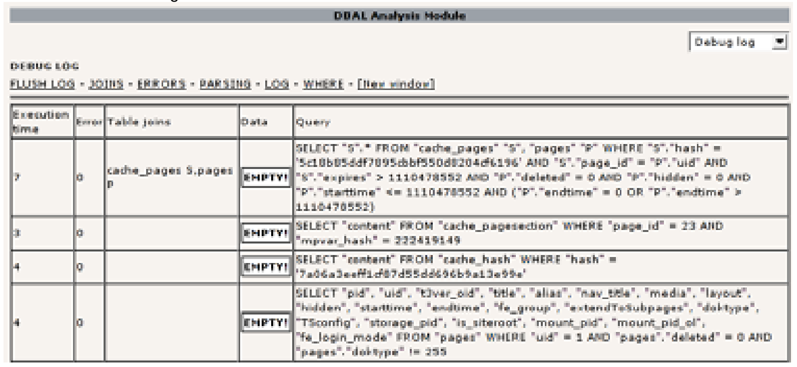
RDBMS	Host	Username	Password	DB Name
MySQL	DB server	Username	Password	Database name
PostgreSQL	DB server	Username	Password	Database name
Oracle	DB server	Username	Password	SID / Instance name Must be entered in *localconf.php* manually!
Firebird	DB server	Username	Password	Full path to the database file, e.g. /tmp/testfb.fdb *Currently not working!*
MS SQL Server (using ODBC)	ODBC DNS	Username	Password	Set to some dummy string!

If your RDBMS is not shown in the list, try with the usual meaning of those parameters first, if that doesn't work, but you figure out how to connect, then please let us know, so we can update this document.

The DBAL Debug backend module

The debug log

When the debug options are enabled queries are logged to a table including various details. You can use the “DBAL debug” backend module to view the log:


DBAL Analysis Module

DEBUG LOG

FLUSH LOG - JOINS - ERRORS - PARSING - LOG - WHERE - [New window]

Execution time	Error	Table joins	Data	Query
7	0	cache_pages S,pages P	EMPTY!	SELECT "S".* FROM "cache_pages" "S", "pages" "P" WHERE "S"."hash" = '5c10b05ddf7095cbbf550d8204df6196' AND "S"."page_id" = "P"."uid" AND "S"."expires" > 1110478552 AND "P"."deleted" = 0 AND "P"."hidden" = 0 AND "P"."starttime" <= 1110478552 AND ("P"."endtime" = 0 OR "P"."endtime" > 1110478552)
3	0		EMPTY!	SELECT "content" FROM "cache_pagesection" WHERE "page_id" = 23 AND "mpvar_hash" = 322419149
4	0		EMPTY!	SELECT "content" FROM "cache_hash" WHERE "hash" = '7a06a3eeff1cf07d55dd696b9a13e99e'
4	0		EMPTY!	SELECT "pid", "uid", "t3ver_oid", "title", "alias", "nav_title", "media", "layout", "hidden", "starttime", "endtime", "fe_group", "extendToSubpages", "doktype", "TSconfig", "storage_pid", "is_siteroot", "mount_pid", "mount_pid_ol", "fe_login_mode" FROM "pages" WHERE "uid" = 1 AND "pages"."deleted" = 0 AND "pages"."doktype" != 255

Above the table you see a few items to click on:

- **FLUSH LOG** clears the logging table.
- **JOINS** shows you the logged table joins, if that feature has been enabled.
- **ERRORS** shows just the entries that caused an error.
- **PARSING** shows the results of the SQL parsing check, if that feature has been enabled.
- **LOG** is what you see when you enter the module.
- **WHERE** shows a log of all WHERE-clauses. This may be used to optimize the database structure or spot performance bottlenecks.

The main log table shows you when how many queries where executed from what script in how much time. If an error occurred, this is noted as well. If you click on the script name, you'll get a list of all queries to help you with debugging.

To aid in debugging problems with null value comparisons '', IS NULL and IS NOT NULL are highlighted in the log and where views in red and green respectively. This indicates no error in itself, so don't worry too much about it.

Viewing cached database information

The DBAL extension makes heavy use of caching for field information. This includes primary keys, auto_increment fields and native/meta types of fields.

To see the cached data select *View cached data* in the DBAL Debug backend module. From there you can clear the cache file, if that seems necessary (it is automatically cleaned whenever the database structure is changed from within TYPO3).

Viewing the configuration

The configuration as it has been defined in *localconf.php* can be checked in the backend by using the Configuration module in the Tools section. Select $TYPO3_CONF_VARS and open the EXTCONF/dbal part of the tree display.

SQL check

Here you can check SQL queries for compliance with the TYPO3 SQL parser. If you enter values in the form fields those are handed to the query building methods and the result is shown. Check of it matches your input to see if the DB subsystem (core and DBAL) can do what you want. For testing inserts, a very simple syntax is used to specify the values to be inserted in the first textarea: Each line is seen as a key/value pair that is exploded at =.

The raw SQL check allows you to enter any query you like and have it parsed and reassembled by TYPO3. If this is successful the input will be shown below. In case of an error (input and output not matching) the query generated by TYPO3 is shown in a red box below the input query.

Installing from scratch with DBAL

Introduction

Installing with DBAL enabled right from the start isn't as easy as it may seem. You need to have two extensions (the DBAL itself and the ADOdb library) installed before even setting up the core system. How this can be done is explained in this section.

Perparing setup

Unpack the TYPO3 source as usual, and unpack a dummy package. Set everything up as explained in the setup documentation, until you come to the point where you are asked to start the install tool – DON'T DO THIS YET!

Install the DBAL and the ADOdb extension in the *“typo3conf/ext/”* folder. Since you cannot (yet) use the extension manager to install it, you need to fetch the sources from somewhere else, e.g. the CVS on the *typo3xdev* project on SourceForge.net.

Open the file *“localconf.php”* and edit it to include the following:

```
$TYPO3_CONF_VARS['EXTCONF']['dbal']['handlerCfg'] = array (
        '_DEFAULT' => array (
            'type' => 'adodb',
            'config' => array(
                'driver' => 'mysql',
            )
        )
    );

$TYPO3_CONF_VARS['EXT']['extList'] .= ',adodb,dbal';
```

Of course you need to adjust the DBAL configuration as you need to, the example above does nothing but route everything through ADOdb inside the DBAL extension. The *extList* is usually not present at this stage, but we need TYPO3 to know about the DBAL extension being present, obviously. And if the extension list is defined, it must include the other default extensions as well, so we don't override the value (coming from config.default.php), but append to it.

Doing the actual setup

Now fire up a browser and visit your new TYPO3 install – you should get redirected to the install tool in 1-2-3 mode. Fill in the database connection parameters and you should get a list of present databases to choose from (if any exist and are accessible to the user entered). If everything went well up to this point, just continue as usual.

If the 1-2-3 more doesn't work, just go to the regular mode and work your way through the setup.

Try not to create a database from within the install tool, this doesn't work anyway (and it probably never will).

Writing DBAL compliant TYPO3 extensions

Introduction

If you want your TYPO3 extensions to be DBAL compliant you might have to rewrite parts of them. The most basic DBAL support is to substitute all direct mysql*() function calls with the wrapper functions found in t3lib_DB accessed through the global object $GLOBALS['TYPO3_DB']. The most radical support is to consistently use the methods in the t3lib_DB class prefixed “exec_” - they will automatically create the proper SQL behind the scenes and execute the queries right away, returning a result pointer/object. This allows the DBAL to handle an ultimate amount of the interaction with the database for you.

SQL standard

When the core of TYPO3 including a number of global extensions were converted to the database wrapper class, t3lib_DB, it was found that the usage of SQL in TYPO3 was luckily quite simple and consistently using the same features. This made it fairly easy to convert the whole application into using the wrapper functions. But it also meant that there was a basis for defining a subset of the MySQL features that are those officially supported by TYPO3.

Yet, this subset is not defined in a document but there exist a class, t3lib_sqlengine, which contains parser functions for SQL and compliance with “TYPO3 sql” is basically defined by whether this class can parse your SQL without errors. (The debug-options /DBAL debug backend module from this extension can be helpful to spot non-compliant SQL.)

This means that TYPO3 now has an official “SQL abstraction language” based on SQL itself and being a subset of the features that MySQL has. Contrary to creating SQL code from a homemade abstraction language there are several advantages in using (a subset of) SQL itself *as* the abstraction language:

- We do not re-invent the wheel by imposing a new “SQL abstraction language” for programmers - they just use simple SQL.
- MySQL (and compatibles) has “native” support and does not need translation (= high speed for our primary database).
- Other databases might need transformation but the overhead can be reduced drastically by simply using the right functions in the DBAL - optionally. And basically such transformation is what would otherwise occur with *any* abstraction language anyways.
- We are able to parse the SQL and validate conformity with the “TYPO3 SQL standard” defined at any time by “t3lib_sqlengine” - and we can always extend it as need arises.

SQL calls

The PHP API for MySQL contains a long list of functions. By the inspection of the TYPO3 core it was found that only quite few of these were used and subsequently only those has made it into the t3lib_DB class (For instance mysql() was typically used to execute a query and subsequently the result was traversed by mysql_fetch_assoc() in 95% of the cases).

The wrapper functions in the class t3lib_DB are now the only functions that a “allowed” to be used for database connectivity and it is believed that these functions is sufficient for all TYPO3 extension programmers. By defining such a limited set of functions we might force some users to change their “habits” of using MySQL/SQL databases but we believe that this is good in another way since we get more consistent code across extensions. If it turns out that some new functions are needed in the wrapper class it must be based on the strength of arguments.

Coding Guidelines

For details on this, please look into the Project Coding Guidelines document - there is a section about this.

Known problems

Database-specific issues

Native handler

The native handler currently does not support importing static data through the install tool or EM, the import will be incomplete in most cases.

PostgreSQL

Currently altering an existing column is not supported, as the underlying ADOdb library needs a full table description to do that, as on PostgreSQL you need to drop and recreate a table to change it's type (this has changed in PostgreSQL 8, but ADOdb doesn't support this yet).

MSSQL

Tests have shown you need to

- enable ANSI quotes (SET QUOTED_IDENTIFIER ON)
- set the max text length in php.ini to a value higher than the default of 4kB
```
; Valid range 0 - 2147483647.  Default = 4096.
mssql.textlimit = 5000000
; Valid range 0 - 2147483647.  Default = 4096.
mssql.textsize = 5000000
```
- Problems with persistent connectiosn were reported, so if you run into trouble, disable them in php.ini

More problems will arise, depending on the setup details. Further fixes and documentation is being worked on.

Results of database comparison operations

Whenever a database structure comparison is done, it is likely that differences are detected, that have no real background. The *unsigned* attribute used with integer fields in MySQL for instance has no equivalent in PostgreSQL. Thus if a change is suggested that would solely try to add this attribute, just ignore it. To make this easier the changes that are usually preselected in the install tool are not, when the DBAL extension is detected.

Similar “errors” will be detected for certain field types and probably keys/indexes. You have to use common sense and your DB background knowledge to move around those issues.

On some databases you might even see keys that should be dropped or created, most of the time this is bogus, too.

Changing of table or field definitions through DBAL

MySQL allows the user to change nearly everything at runtime, you can change field types, constraints, defaults and more on a field and MySQL handles data conversion and similar tasks transparently. This does work very differently with many other DB systems, some things do not work at all. Since we rely on existing abstraction libraries, we are bound to what they offer. There may be cases in which it simply is impossible to do a needed change to the database automatically.

It is even problematic to add fields to existing tables, although this seems like a rather simple thing to do. But, alas, it is not.

It must be noted that e.g. the recent version 8.0 of PostgreSQL was a huge step forward in that direction, as it allows a lot more of those operations in an easy way, than earlier versions. So if you want to run TYPO3 on PostgreSQL, use version 8.0 or later, of possible.

Database Check in Backend

The *Advanced Query* tool in the *Full Search* of the *DB Check* in the backend doesn't “speak DBAL” yet, so it won't work with anything but MySQL until this has been worked on.

Sequences collide with imported data

If you have imported data into a table that is not running on MySQL, you may see error messages when inserting a new record (see e.g. bug #2106 on http://bugs.typo3.org/). This can be avoided by setting the sequence start to a number higher then the highest ID already in use in all the tables for that handler. See above for the configuration syntax.

Some field names are too long

Oracle for example has a restriction on table and field names, they can only be 30 bytes long. Some fields, especially of extensions adding fields to existing tables may violate that restriction. A way to work around this is to configure a field name mapping for those long names onto a shorter one before creating them (through the EM or install tool).

Creating a database from within the install tool

Because of the differences between the way RDBMS create databases, this isn't possible and probably never will be.

FAQ

How to import DB dumps with the install tool

If you want to move a site from a MySQL setup to a DBAL setup with another database, you can either use the Import/Export extension (impexp) or create a dump to be imported by the install tool. For this to work a few hints should be followed:

1. Create the MySQL using mysqldump with thses options:
   --compatible=mysql40 --complete-insert --skip-opt --skip-quote-names --skip-comments

   Empty tables as well as cache_*, session_* and maybe the sys_log and sys_history (according to your preference) can be left out of the dump.
2. Make sure the dump has no backticks around table or field names (during testing there still were some, despite the options above being used). Don't do an unconditional replace over the whole dump, though, as there may be backticks inside the actual data. Be careful!
3. Put the dump in typo3conf/ and import it using the install tool.
4. Now use the "Update required tables COMPARE" in the install tool to create the remaining missing or changed tables and fields.

Can I put the cache tables in a different database?

Yes, you can map the cache tables somewhere else. There is one caveat, though – if you put the cache_pages table into a different database than the pages table the FE will throw an error. The tslib_fe class uses a join over those two tables, which cannot work. Never.

You can do two things to work around this: Use the file-based caching that is available since TYPO3 4.0.0 or apply the patch class.tslib_fe.php.diff found in the doc/ directory of the DBAL extension.

What happens to the table definitions in MySQL format?

Right, all table definitions in TYPO3 and it's extensions come in a format compatible to what mysqldump produces. Still, the DBAL can handle this, and here are the rules that apply when the ADOdb handler is used:

- MySQL field types are mapped onto meta types, taken from ADOdb, but adapted to better fit the needs of TYPO3. From those field types the actual types for the target database are generated with the ADOdb library.
- If we need to map the actual field type in the database back onto a MySQL type, we use the same system backwards. This explains why most field type comparisons don't match exactly. An example: aTINYINT is mapped to the meta type I2, this is mapped to some DB-specific type. Later the actual type is mapped back to I2 and then to SMALLINT. Bang, the types do not match.

- If a field has no default value assigned in the dump, it is assigned either 0 or an empty string as default (depending on it's type). This is done to fake the implicit default values MySQL assigns to fields that have no explicit default.
- The UNSIGNED attribute for integer fields is dropped for all databases except MySQL when using the ADOdb handler, as it is MySQL specific.
- The AUTO_INCREMENT attribute is never used for the ADOdb handler, (emulated) sequences are used instead.
- If a field is declared NOT NULL in the MySQL dump, this will be changed to allow NULL if running on Oracle..

Notice: For the native handler no conversion is done.

Why drop NOT NULL constraints and add default values?

All the NOT NULL fields in the database dumps only work on MySQL, because *all* fields *always* have a default value in MySQL, even if none is given explicitly, if they are NOT NULL. Yes, MySQL always assigns a default value – there are no fields without a default value.

In other databases this is not the case, so that any NOT NULL field not being assigned a value during an INSERT triggers an error. This in itself is perfectly fine, were TYPO3 not to omit a lot of fields in a lot INSERT queries. This is why a default value is always added – it avoids errors during INSERT queries.

Now, we still drop the NOT NULL when running on Oracle, why is that the case? Oracle treats an empty string as being null, so you cannot insert an empty string into a field being NOT NULL. Furthermore, having an empty string as default value is the same as having null as default value. The bottom line is: it doesn't work having a NOT NULL field with NULL as default, as you'd get errors during INSERT. Now go and complain to the folks over at Oracle...

Bugs

Please use the bug tracker at http://bugs.typo3.org/ to submit any bugs you find in this extension (or near it) to the project *tx_dbal*.

If the DBAL doesn't work for you the way it should, but you need it to work to meet a deadline, consider paying for the fixes – just get in touch with Karsten Dambekalns.

To-Do list

This list has been radically edited for version 0.9.9 after being the unchanged TODO.txt in the doc directory for a rather long time. It is now much shorter but still needs some points to be checked and added...

New Features?

- user processing hooks implemented on tables/fields etc.
- readOnly control on tables/handlers
- persistent connections configured as an option per handler?

General TODO:

- Implement and test other databases with ADOdb and PEAR
- Create any amount of management needed
- Create documentation for other databases
- Support for people implementing DBAL (I imagine there is a contact person per. DBMS implemented)

Oracle

- Insert a "" quoted string in an integer/number field - even if the value inside is a number! - so will we have to make lookups on field type to determine this?
- Insert a string longer than the size of the field in the database (MySQL just silently accepts this...) - so will we have to evaluate all values in update/insert queries first?
- does not allow us to CHANGE existing fields into something else - only create new fields, otherwise we must export/import database.
- a quoted value cannot be inserted into an integerfield!!!