Database Independent Abstraction Layer for C

libdbi Programmer's Guide

David A. Parker
Neon Goat Productions

david@neongoat.com

Database Independent Abstraction Layer for C: libdbi Programmer's Guide

by David A. Parker

Document revision: $Id: programmers-guide.sgml,v 1.7 2001/07/19 08:15:25 dap24 Exp $ Edition
Published June 11, 2001

libdbi implements a database-independent abstraction layer in C, similar to the DBI/DBD layer in Perl. Writing one generic set of code, programmers can leverage the power of multiple databases and multiple simultaneous database connections by using this framework.

Table of Contents

Chapter 1. Introduction

1.1. Description

libdbi provides application developers with a database independent abstraction layer for C. It handles the database-specific implementations for each type of database, so that you can use the same exact code with any type of database server that libdbi supports. You can initiate and use multiple database connections simultaneously, regardless of the types of database servers you are connecting to. The plugin architecture allows for new database drivers to be easily added dynamically by a third party.

1.2. libdbi Concepts and Terminology

In this guide, the terms “user” and “programmer” are used interchangably, since the target audience is the software developer using libdbi in his program. The libdbi architecture provides serveral “plugins”, one for each type of database server. All plugins are loaded into memory upon libdbi initiallization and are made available to the programmer. Once a plugin is *instantiated*, it represents a distinct database session and is called a “driver”. Multiple drivers may exist for a single plugin, and all will function independently of each other. A star character (*) represents a wildcard matching any letters. For example, “dbi_driver_*” would represent all functions beginning with “dbi_driver_”.

1.3. Modifications and redistribution of libdbi

1.4. Contact Info

Please email us with any bugs, ideas, feature requests, or questions. The libdbi website has the latest version of this documentation and the libdbi software, as well as a central database of third-party plugins.

- http://libdbi.sourceforge.net
- David Parker <david@neongoat.com>
- Mark Tobenkin <mark@brentwoodradio.com>

Chapter 2. libdbi in a Nutshell (Quickstart Guide)

2.1. Quick Overview

libdbi uses a plugin system that allows various databases to be supported simultaneously, and can dynamically load or unload plugins that are supplied by libdbi or a third party. The library is initiallized by calling dbi_initialize and a plugin instance (a driver) is started by calling either dbi_driver_new or both dbi_plugin_open and dbi_driver_open.

The driver's options (username, password, hostname, etc.) are set with dbi_driver_set_option and dbi_driver_set_option_numeric. Once all options are set, dbi_driver_connect will connect to the database, waiting to handle a dbi_driver_query. After a successful query, you can retrieve rows with dbi_result_first_row, dbi_result_last_row, dbi_result_prev_row, dbi_result_next_row, and dbi_result_seek_row.

There are two methods for fetching field data, and two ways to perform each method. You can either "pull" the data from DBI using the `dbi_result_get_*` family of functions, or have DBI automatically "push" the data into predefined variables with the `dbi_result_bind_*` family of functions.

Pulling the data from the database can be done with one of the "get" functions such as dbi_result_get_long or dbi_result_get_string, which simply return the data in the field you asked for. You can also get more than one field at a time with dbi_result_get_fields, which uses a printf-like syntax.

If you want DBI to automatically fill your program's variables with field values whenever a new row is fetched, you can "bind" fields to your variables. Bindings are set up with dbi_result_bind_long, dbi_result_bind_string, and the rest of the bind family of functions. Like the associated "get" function, you can set up multiple bindings at once with the dbi_result_bind_fields function.

Caveats:

- For fields holding integers (not fractional numbers), DBI differentiates between signed and unsigned variables. By default, DBI returns signed values. If you want an unsigned value, prepend a "u" to the name of the target type. For example, dbi_result_bind_short becomes dbi_result_bind_ushort.
- You must set up any bindings AFTER a successful query but BEFORE you fetch any rows. Even if you are using field bindings, you can still use the dbi_result_get_* functions as usual. (actually, I lied... setting up a binding should theoretically work at any time, but don't plan on this behavior in future versions)
- All string and binary data returned or bound from DBI is READ-ONLY. If you want your own local copy that can be modified at will, use dbi_result_get_string_copy, dbi_result_get_binary_copy, dbi_result_bind_string_copy, or dbi_result_bind_binary_copy. You will be responsible for freeing the memory allocated by these functions.

dbi_result_next_row and the other row-seeking functions will return zero when there are no more rows available. Before the next database operation is performed, you must call dbi_result_free. Before the program terminates, the driver must be disconnected and unloaded with dbi_driver_close and libdbi must be unloaded with dbi_shutdown.

2.2. Generic Example Program

```
#include <stdio.h>
#include <dbi/dbi.h>
```

```
int main() {
    dbi_driver driver;
    dbi_result result;

    double threshold = 4.333333;
    unsigned int idnumber;
    const char *fullname;

    dbi_initialize(NULL);
    driver = dbi_driver_new("mysql");

    dbi_driver_set_option(driver, "host", "localhost");
    dbi_driver_set_option(driver, "username", "chug");
    dbi_driver_set_option(driver, "password", "dIP!");
    dbi_driver_set_option(driver, "dbname", "db_name");

    dbi_driver_connect(driver);
    result = dbi_driver_query(driver, "SELECT id, name FROM coders"
                                      "WHERE hours_of_sleep > %0.2f", threshold);

    while (dbi_result_next_row(result)) {
        idnumber = dbi_result_get_long(result, "id");
        fullname = dbi_result_get_string(result, "name");
        printf("%i. %s\n", idnumber, fullname);
    }

    dbi_result_free(result);
    dbi_driver_close(driver);
    dbi_shutdown();

    return 0;
}
```

Compile with: gcc -ldl -ldbi -o foo foo.c

Of course, a complete program should be checking for errors. This example omits error-checking for the sake of clarity. There are also other ways to retrieve data after a successful query. Keep reading on to see the rest.

Chapter 3. libdbi API Reference

3.1. Core Library Functions

3.1.1. dbi_initialize

```
int dbi_initialize(const char *plugindir)
```

Locates all available shared modules (plugins) and loads them into memory.

Arguments

`plugindir`: The directory to search for plugins. If NULL, DBI_PLUGIN_DIR (defined at compile time) will be used instead.

Returns

The number of plugins successfully loaded, or -1 if there was an error.

3.1.2. dbi_shutdown

```
void dbi_shutdown()
```

Frees all loaded plugins and terminates the DBI system. You should close each driver you opened before shutting down, but libdbi will clean up after you if you don't.

3.1.3. dbi_version

```
const char *dbi_version()
```

Requests the version of libdbi. The calling program must not attempt to free the returned string.

Returns

A string containing the library's name and version.

3.2. Plugin Infrastructure

3.2.1. dbi_plugin_list

```
dbi_plugin dbi_plugin_list(dbi_plugin Current)
```

Enumerates all loaded plugins. If Current is NULL, the first available plugin will be returned. If Current is a valid plugin, the next available plugin will be returned.

Arguments

`Current`: The current plugin in the list of plugins.

Returns

The next available plugin, or NULL if there is an error or no more are available.

3.2.2. dbi_plugin_open

```
dbi_plugin dbi_plugin_open(const char *name)
```

Locate the plugin with the specified name.

Arguments

`name`: The name of the plugin to open.

Returns

The requested plugin, or NULL if there is an error or it is not found.

3.2.3. dbi_plugin_is_reserved_word

```
int dbi_plugin_is_reserved_word(dbi_plugin Plugin, const char *word)
```

Looks for the specified word in the list of reserved words. The result of this function may vary between databases. Case does not matter.

Arguments

`Plugin`: The target plugin.

`word`: The word to check against the reserved word list.

Returns

-1 if an error occurs, 0 if the word is not reserved, 1 otherwise.

3.2.4. dbi_plugin_specific_function

```
void *dbi_plugin_specific_function(dbi_plugin Plugin, const char *name)
```

Returns a function pointer to the specifed custom function. This can be used to access database-specific functionality, but it will restrict your code to one particular database, lessening the benefits of using libdbi.

Arguments

`Plugin`: The target plugin.

`name`: The name of the custom function.

Returns

If the custom function is found, a pointer to that function. If not, returns NULL.

3.2.5. dbi_plugin_quote_string

```
int dbi_plugin_quote_string(dbi_plugin Plugin, char **orig)
```

Encloses the target string in the types of quotes that the database expects, and escapes any special characters. The original string will be freed and will point to a newly allocated one (which you still must free on your own).

Arguments

`Plugin`: The target plugin.

`orig`: A pointer to the string to quote and escape.

Returns

The new string's length.

3.2.6. Plugin Information

3.2.6.1. dbi_plugin_get_name

```
const char *dbi_plugin_get_name(dbi_plugin Plugin)
```

Requests the name of the specified plugin. The calling program must not attempt to free the returned string.

Arguments

`Plugin`: The target plugin.

Returns

A string containing the plugin's name.

3.2.6.2. dbi_plugin_get_filename

```
const char *dbi_plugin_get_filename(dbi_plugin Plugin)
```

Requests the filename of the specified plugin. The calling program must not attempt to free the returned string.

Arguments

`Plugin`: The target plugin.

Returns

A string containing the plugin's full path and file name.

3.2.6.3. dbi_plugin_get_description

```
const char *dbi_plugin_get_description(dbi_plugin Plugin)
```

Requests a description of the specified plugin. The calling program must not attempt to free the returned string.

Arguments

`Plugin`: The target plugin.

Returns

A string containing the plugin's description. It will be one or two short sentences with no newlines.

3.2.6.4. dbi_plugin_get_maintainer

```
const char *dbi_plugin_get_maintainer(dbi_plugin Plugin)
```

Requests the maintainer of the specified plugin. The calling program must not attempt to free the returned string.

Arguments

`Plugin`: The target plugin.

Returns

A string containing the plugin maintainer's full name and email address.

3.2.6.5. dbi_plugin_get_url

```
const char *dbi_plugin_get_url(dbi_plugin Plugin)
```

Requests the maintainer's URL for the specified plugin. This is useful for plugins maintained by a third party. The calling program must not attempt to free the returned string.

Arguments

`Plugin`: The target plugin.

3.2.6.6. dbi_plugin_get_version

```
const char *dbi_plugin_get_version(dbi_plugin Plugin)
```

Requests the version of the specified plugin. The calling program must not attempt to free the returned string.

Arguments

`Plugin`: The target plugin.

Returns

A string containing the plugin's version.

3.2.6.7. dbi_plugin_get_date_compiled

```
const char *dbi_plugin_get_date_compiled(dbi_plugin Plugin)
```

Requests the compilation date of the specified plugin. The calling program must not attempt to free the returned string.

Arguments

`Plugin`: The target plugin.

Returns

A string containing the date the plugin was compiled.

3.3. Driver Infrastructure

3.3.1. dbi_driver_new

```
dbi_driver dbi_driver_new(const char *name)
```

Creates a driver instance of the plugin specified by "name". This is a shortcut for calling dbi_plugin_open() and passing the result to dbi_driver_open().

Arguments

`name`: The name of the desired plugin.

Returns

A driver instance of the specified plugin, or NULL if there was an error.

3.3.2. dbi_driver_open

```
dbi_driver dbi_driver_open(dbi_plugin Plugin)
```

Creates a driver instance of the specified plugin. This driver can be used to perform queries and set options.

Arguments

`Plugin`: The target plugin.

Returns

A driver instance of the specified plugin, or NULL if there was an error.

3.3.3. dbi_driver_get_plugin

```
dbi_plugin dbi_driver_get_plugin(dbi_driver Driver)
```

Returns the plugin type of the specified driver.

Arguments

`Driver`: The target driver.

Returns

The plugin type of the target driver.

3.3.4. dbi_driver_set_option

```
int dbi_driver_set_option(dbi_driver Driver, const char *key, char *value)
```

Sets a specified driver option to a string value.

Arguments

`Driver`: The target driver.

`key`: The name of the target setting. Must only contain alphanumerics and the underscore character.

`value`: The string value of the target setting.

Returns

-1 on error, 0 on success.

3.3.5. dbi_driver_set_option_numeric

```
int dbi_driver_set_option_numeric(dbi_driver Driver, const char *key, int value)
```

Sets a specified driver option to a numeric value.

Arguments

`Driver`: The target driver.

`key`: The name of the target setting. Must only contain alphanumerics and the underscore character.

`value`: The numeric value of the target setting.

Returns

-1 on error, 0 on success.

3.3.6. dbi_driver_get_option

```
const char *dbi_driver_get_option(dbi_driver Driver, const char *key)
```

Retrieves the string value of the specified option set for a driver.

Arguments

`Driver`: The target driver.

`key`: The name of the target setting.

Returns

A read-only string with the setting, or NULL if it is not available.

3.3.7. dbi_driver_get_option_numeric

```
int dbi_driver_get_option_numeric(dbi_driver Driver, const char *key)
```

Retrieves the integer value of the specified option set for a driver.

Arguments

`Driver`: The target driver.

`key`: The name of the target setting.

Returns

The value of the setting, or -1 if it is not available.

3.3.8. dbi_driver_get_option_list

```
const char *dbi_driver_get_option_list(dbi_driver Driver, const char *current)
```

Enumerates the list of available options for a driver. If current is NULL, the first available option will be returned. If current is a valid option name, the next available option will be returned.

Arguments

`Driver`: The target driver.

`current`: The key name of the target option.

Returns

The key name of the next option, or NULL if there was an error or there are no more options.

3.3.9. dbi_driver_clear_option

```
void dbi_driver_clear_option(dbi_driver Driver, const char *key)
```

Removes the target option setting from a driver.

Arguments

`Driver`: The target driver.

`key`: The name of the target setting.

3.3.10. dbi_driver_clear_options

```
void dbi_driver_clear_options(dbi_driver Driver)
```

Removes all option settings from a driver.

Arguments

`Driver`: The target driver.

3.3.11. dbi_driver_close

```
void dbi_driver_close(dbi_driver Driver)
```

Disconnects the specified driver connection from the database and cleans up the driver session.

Arguments

`Driver`: The target driver.

3.3.12. Error Handling

3.3.12.1. dbi_driver_error

```
int dbi_driver_error(dbi_driver Driver, char **errmsg_dest)
```

Generates a formatted message with the error number and description resulting from the previous database operation, copying the message into the specified string.

Arguments

`Driver`: The target driver.

`errmsg_dest`: The target string pointer, which will point to the error message. If NULL, no error message will be created, but the error number will still be returned.

Returns

The error number of the most recent database operation if it resulted in an error. If not, this will return -1.

3.3.12.2. dbi_driver_error_handler

```
void dbi_driver_error_handler(dbi_driver Driver, void *function, void *user_argument)
```

Registers an error handler callback to be triggered whenever the database encounters an error. The callback function should perform as little work as possible, since the state in which it is called can be uncertain. The actual function declaration must accept two parameters:

- dbi_driver_t *driver: a pointer to the driver that triggered the error, from which dbi_error() can be called, and
- void *user_argument: a pointer to whatever data (if any) was registered along with the handler.

To remove the error handler callback, specify NULL as the function and user_argument.

Arguments

`Driver`: The target driver.

`function`: A pointer to the function to call when the error handler should be triggered.

`user_argument`: Any data to pass along to the function when it is triggered. Set to NULL if unused.

3.4. SQL and Database Infrastructure

3.4.1. dbi_driver_connect

```
int dbi_driver_connect(dbi_driver Driver)
```

Connects to the database using the options (host, username, password, port, (etc.) set with dbi_set_option() and dbi_set_option_numeric(). See the documentation for each specific database plugin for the options it recognizes and requires.

Arguments

`Driver`: The target driver.

Returns

-1 on failure, zero on success.

3.4.2. dbi_driver_get_db_list

```
dbi_result dbi_driver_get_db_list(dbi_driver Driver)
```

Queries the list of available databases on the server.

Arguments

`Driver`: The target driver.

Returns

A query result object, which will contain a field named "dbname" from which the standard row/field fetching functions can be used.

3.4.3. dbi_driver_get_table_list

```
dbi_result dbi_driver_get_table_list(dbi_driver Driver, const char *db)
```

Queries the list of available tables in a particular database.

Arguments

`Driver`: The target driver.

`db`: The target database name.

Returns

A query result object, which will contain a field named "tablename" from which the standard row/field fetching functions can be used.

3.4.4. dbi_driver_query

```
dbi_result dbi_driver_query(dbi_driver Driver, const char *formatstr, ...)
```

Execute the specified SQL query statement.

Arguments

`Driver`: The target driver.

`formatstr`: The format string for the SQL statement. It uses the same format as printf().

`ARG`: (...) Any variables that correspond to the printf-like format string.

Returns

A query result object, or NULL if there was an error.

3.4.5. dbi_driver_select_db

```
int dbi_driver_select_db(dbi_driver Driver, const char *db)
```

Switches to a different database on the server.

Arguments

`Driver`: The target driver.

`db`: The target database name.

Returns

-1 on failure, zero on success.

3.4.6. dbi_result_get_driver

```
dbi_driver dbi_result_get_driver(dbi_result Result)
```

Returns the driver belonging to the specified result object.

Arguments

`Result`: The target query result.

Returns

The driver belonging to the target query result.

3.4.7. dbi_result_free

```
int dbi_result_free(dbi_result Result)
```

Frees the result's query, disables all stored field bindings, and releases internally stored variables.

Arguments

`Result`: The target query result.

Returns

-1 on failure, zero on success.

3.4.8. dbi_result_seek_row

```
int dbi_result_seek_row(dbi_result Result, unsigned int row)
```

Jump to a specific row in a result set.

Arguments

`Result`: The target query result.

`row`: The ordinal number of the row to seek to. The first row is at position 1, not zero.

Returns

The row number that was fetched, or 0 if there is an error.

3.4.9. dbi_result_first_row

```
int dbi_result_first_row(dbi_result Result)
```

Jump to the first row in a result set.

Arguments

`Result`: The target query result.

Returns

The row number that was fetched, or 0 if there is an error.

3.4.10. dbi_result_last_row

```
int dbi_result_last_row(dbi_result Result)
```

Jump to the last row in a result set.

Arguments

`Result`: The target query result.

Returns

The row number that was fetched, or 0 if there is an error.

3.4.11. dbi_result_prev_row

```
int dbi_result_prev_row(dbi_result Result)
```

Jump to the previous row in a result set.

Arguments

`Result`: The target query result.

Returns

The row number that was fetched, or 0 if there is an error.

3.4.12. dbi_result_next_row

```
int dbi_result_next_row(dbi_result Result)
```

Jump to the next row in a result set.

Arguments

`Result`: The target query result.

Returns

The row number that was fetched, or 0 if there is an error.

3.4.13. dbi_result_get_numrows

```
unsigned int dbi_result_get_numrows(dbi_result Result)
```

Returns the number of rows in the specified result set.

Arguments

`Result`: The target query result.

Returns

The number of rows in the result set.

3.4.14. dbi_result_get_numrows_affected

```
unsigned int dbi_result_get_numrows_affected(dbi_result Result)
```

Returns the number of rows in the specified result set that were actually modified. Note that not all database servers support this, in which case it will always be zero. See the documentation for each specific plugin for details.

Arguments

`Result`: The target query result.

Returns

The number of modified rows in the result set.

3.5. Retrieving field data

3.5.1. dbi_result_get_field_size

```
unsigned int dbi_result_get_field_size(dbi_result Result, const char *fieldname)
```

Returns the size in bytes of the value stored in the specified field. This is especially useful for string and binary data fields, which may have a dynamic size.

Arguments

`Result`: The target query result.

`fieldname`: The name of the target field.

Returns

The size in bytes of the target field data.

3.5.2. dbi_result_get_field_size_idx

```
unsigned int dbi_result_get_field_size_idx(dbi_result Result, unsigned int idx)
```

Returns the size in bytes of the value stored in the specified field. This is especially useful for string and binary data fields, which may have a dynamic size.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The size in bytes of the target field data.

3.5.3. dbi_result_get_field_length

```
unsigned int dbi_result_get_field_length(dbi_result Result, const char *fieldname)
```

Returns the length in bytes of the value stored in the specified field. This is always one less than the size, and is probably only useful for fields containing strings.

Arguments

`Result`: The target query result.

`fieldname`: The name of the target field.

Returns

The length in bytes of the target field data.

3.5.4. dbi_result_get_field_length_idx

```
unsigned int dbi_result_get_field_length_idx(dbi_result Result, unsigned int idx)
```

Returns the length in bytes of the value stored in the specified field. This is always one less than the size, and is probably only useful for fields containing strings.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The length in bytes of the target field data.

3.5.5. dbi_result_get_field_idx

```
int dbi_result_get_field_idx(dbi_result Result, const char *fieldname)
```

Given a field’s name, return that field’s numeric index.

Arguments

`Result`: The target query result.

`fieldname`: The name of the target field.

Returns

The index (starting at 1) of the target field.

3.5.6. dbi_result_get_field_name

```
const char *dbi_result_get_field_name(dbi_result Result, unsigned int idx)
```

Given a field's numeric index, return that field's name.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The target field's name.

3.5.7. dbi_result_get_numfields

```
unsigned int dbi_result_get_numfields(dbi_result Result)
```

Returns the number of fields in the query result.

Arguments

`Result`: The target query result.

Returns

The number of fields in the query result.

3.5.8. dbi_result_get_field_type

```
unsigned short dbi_result_get_field_type(dbi_result Result, const char *fieldname)
```

Returns the target field's data type. The constants returned by this function are defined in dbi.h with the prefix "DBI_TYPE_".

Arguments

`Result`: The target query result.

`fieldname`: The target field's name.

Returns

The target field's data type.

3.5.9. dbi_result_get_field_type_idx

```
unsigned short dbi_result_get_field_type_idx(dbi_result Result, unsigned int idx)
```

Returns the target field's data type. The constants returned by this function are defined in dbi.h with the prefix "DBI_TYPE_".

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The target field's data type.

3.5.10. dbi_result_get_field_attrib

```
unsigned long dbi_result_get_field_attrib(dbi_result Result, const char *field-
name, unsigned long attribmin, unsigned long attribmax)
```

Returns the target field's data type attributes in the specified range. The constants returned by this function are defined in dbi.h with the prefix "DBI_", followed by the name of the field's datatype.

Arguments

`Result`: The target query result.

`fieldname`: The target field's name.

`attribmin`: The first attribute value in the range of attributes to extract.

`attribmax`: The last attribute value in the range of attributes to extract. This may be the same as attribmin if you are only trying to extract a single attribute value.

Returns

The target field's requested attribute range.

3.5.11. dbi_result_get_field_attrib_idx

```
unsigned long dbi_result_get_field_attrib_idx(dbi_result Result, unsigned int idx, un-
signed long attribmin, unsigned long attribmax)
```

Returns the target field's data type attributes in the specified range. The constants returned by this function are defined in dbi.h with the prefix "DBI_", followed by the name of the field's datatype.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

`attribmin`: The first attribute value in the range of attributes to extract.

`attribmax`: The last attribute value in the range of attributes to extract. This may be the same as attribmin if you are only trying to extract a single attribute value.

Returns

The target field's requested attribute range.

3.5.12. dbi_result_get_field_attribs

```
unsigned long dbi_result_get_field_attribs(dbi_result Result, const char *fieldname)
```

Returns the target field's data type attributes. The constants returned by this function are defined in dbi.h with the prefix "DBI_", followed by the name of the field's datatype.

Arguments

`Result`: The target query result.

`fieldname`: The target field's name.

Returns

The target field's attributes.

3.5.13. dbi_result_get_field_attribs_idx

```
unsigned long dbi_result_get_field_attribs_idx(dbi_result Result, unsigned int idx)
```

Returns the target field's data type attributes. The constants returned by this function are defined in dbi.h with the prefix "DBI_", followed by the name of the field's datatype.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The target field's attributes.

3.5.14. dbi_result_get_fields

```
int dbi_result_get_fields(dbi_result Result, const char *format, ...)
```

Fetch multiple fields from the current result set, using a printf-like syntax. The formatter string specified field names and types, and each field's associated destination variable is passed as an argument following the format string. Fields in the formatter string are separated by spaces, and follow the format `"a.%b"`, where "a" is the name of the field, and "b" is the field type specifier. Make sure you pass the destination variables' memory addresses by prepending the & operator to each variable's name.

Field type specifiers:

- `%c / %uc`: A signed/unsigned character
- `%h / %uh`: A signed/unsigned short integer
- `%l / %ul`: A signed/unsigned long integer
- `%i / %ui`: A signed/unsigned long integer
- `%L / %uL`: A signed/unsigned long long integer
- `%f`: A floating point number
- `%d`: A double-precision number
- `%s`: A read-only string
- `%S`: A local copy of a string (must be freed by program)
- `%b`: A read-only pointer to binary data
- `%B`: A local copy of binary data (must be freed by program)
- `%t`: A read-only string representing a SET
- `%e`: A read-only string representing an ENUM
- `%m`: A time_t value representing a DATE and/or TIME

Example usage: `dbi_result_get_fields(result, "idnum.%ul lastname.%s", &id_number, &name)`

Arguments

`Result`: The target query result.

`format`: The field format string as described above.

`ARG`: (...) Pointers to the destination variables corresponding with each field in the format string.

Returns

The number of fields fetched, or -1 if there was an error. If an invalid field name was specified it will not cause -1 to be returned, and the other fetched fields will work as usual.

3.5.15. dbi_result_bind_fields

```
int dbi_result_bind_fields(dbi_result Result, const char *format, ...)
```

Bind multiple fields in the current result set, using a printf-like syntax. See dbi_result_get_fields for a detailed explanation of the syntax.

Arguments

`Result`: The target query result.

`format`: The field format string as described above.

`ARG`: (...) Pointers to the destination variables corresponding with each field in the format string.

Returns

The number of field binding set up, or -1 if there was an error.

3.5.16. dbi_result_get_char

```
signed char dbi_result_get_char(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the speficied field, which contains a character.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.17. dbi_result_get_uchar

```
unsigned char dbi_result_get_uchar(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the speficied field, which contains an unsigned character.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.18. dbi_result_get_short

```
short dbi_result_get_short(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the speficied field, which contains a short integer.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.19. dbi_result_get_ushort

```
unsigned short dbi_result_get_ushort(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the speficied field, which contains an unsigned short integer.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.20. dbi_result_get_long

```
long dbi_result_get_long(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the speficied field, which contains a long integer.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.21. dbi_result_get_ulong

```
unsigned long dbi_result_get_ulong(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the speficied field, which contains an unsigned long integer.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.22. dbi_result_get_longlong

```
long long dbi_result_get_longlong(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the speficied field, which contains a long long integer.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.23. dbi_result_get_ulonglong

```
unsigned long long dbi_result_get_ulonglong(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the speficied field, which contains an unsigned long long integer.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.24. dbi_result_get_float

```
float dbi_result_get_float(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the speficied field, which contains a floating-point number.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.25. dbi_result_get_double

```
double dbi_result_get_double(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the speficied field, which contains a double-precision fractional number.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.26. dbi_result_get_string

```
const char *dbi_result_get_string(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the speficied field, which contains a string. The string may not be modified, and may not necessairly persist between row fetches.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.27. dbi_result_get_binary

```
const unsigned char *dbi_result_get_binary(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the speficied field, which contains binary BLOB data. The data may not be modified, and may not necessarily persist between row fetches.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.28. dbi_result_get_string_copy

```
char *dbi_result_get_string_copy(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the speficied field, which contains a string. The newly allocated string may be modified by the host program, but the program is responsible for freeing the string.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.29. dbi_result_get_binary_copy

```
unsigned char *dbi_result_get_binary_copy(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the speficied field, which contains binary BLOB data. The newly allocated memory may be modified by the host program, but the program is responsible for freeing the data.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.30. dbi_result_get_enum

```
const char *dbi_result_get_enum(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the speficied field, which contains an ENUM (which will be represented as a read-only string).

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.31. dbi_result_get_set

```
const char *dbi_result_get_set(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the speficied field, which contains a SET (which will be represented as a read-only string).

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.32. dbi_result_get_datetime

```
time_t dbi_result_get_datetime(dbi_result Result, const char *fieldname)
```

Fetch the data stored in the specified field, which contains a DATE and/or TIME value.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to fetch.

Returns

The data stored in the specified field.

3.5.33. dbi_result_bind_char

```
int dbi_result_bind_char(dbi_result Result, const char *fieldname, char *bindto)
```

Bind the specified variable to the specified field, which holds a character.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.34. dbi_result_bind_uchar

```
int dbi_result_bind_uchar(dbi_result Result, const char *fieldname, un-
signed char *bindto)
```

Bind the specified variable to the specified field, which holds an unsigned character.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.35. dbi_result_bind_short

```
int dbi_result_bind_short(dbi_result Result, const char *fieldname, short *bindto)
```

Bind the specified variable to the specified field, which holds a short integer.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.36. dbi_result_bind_ushort

```
int dbi_result_bind_ushort(dbi_result Result, const char *fieldname, un-
signed short *bindto)
```

Bind the specified variable to the specified field, which holds an unsigned short integer.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.37. dbi_result_bind_long

```
int dbi_result_bind_long(dbi_result Result, const char *fieldname, long *bindto)
```

Bind the specified variable to the specified field, which holds a long integer.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.38. dbi_result_bind_ulong

```
int dbi_result_bind_ulong(dbi_result Result, const char *fieldname, un-
signed long *bindto)
```

Bind the specified variable to the specified field, which holds an unsigned long integer.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.39. dbi_result_bind_longlong

```
int dbi_result_bind_longlong(dbi_result Result, const char *field-
name, long long *bindto)
```

Bind the specified variable to the specified field, which holds a long long integer.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.40. dbi_result_bind_ulonglong

```
int dbi_result_bind_ulonglong(dbi_result Result, const char *fieldname, un-
signed long long *bindto)
```

Bind the specified variable to the specified field, which holds an unsigned long long integer.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.41. dbi_result_bind_float

```
int dbi_result_bind_float(dbi_result Result, const char *fieldname, float *bindto)
```

Bind the specified variable to the specified field, which holds a floating-point number.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.42. dbi_result_bind_double

```
int dbi_result_bind_double(dbi_result Result, const char *fieldname, double *bindto)
```

Bind the specified variable to the specified field, which holds a double-precision fractional number.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.43. dbi_result_bind_string

```
int dbi_result_bind_string(dbi_result Result, const char *field-
name, const char **bindto)
```

Bind the specified variable to the specified field, which holds a string. The string must not be modified.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.44. dbi_result_bind_binary

```
int dbi_result_bind_binary(dbi_result Result, const char *fieldname, const un-
signed char **bindto)
```

Bind the specified variable to the specified field, which holds binary BLOB data. The data must not be modified.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.45. dbi_result_bind_string_copy

```
int dbi_result_bind_string_copy(dbi_result Result, const char *field-
name, char **bindto)
```

Bind the specified variable to the specified field, which holds a string. The newly allocated string may be modified by the host program, but the program is responsible for freeing the string.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.46. dbi_result_bind_binary_copy

```
int dbi_result_bind_binary_copy(dbi_result Result, const char *fieldname, un-
signed char **bindto)
```

Bind the specified variable to the specified field, which holds binary BLOB data. The newly allocated data may be modified by the host program, but the program is responsible for freeing the data.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.47. dbi_result_bind_enum

```
int dbi_result_bind_enum(dbi_result Result, const char *field-
name, const char **bindto)
```

Bind the specified variable to the specified field, which holds an ENUM (which is represented as a read-only string).

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.48. dbi_result_bind_set

```
int dbi_result_bind_set(dbi_result Result, const char *fieldname, const char **bindto)
```

Bind the specified variable to the specified field, which holds a SET (which is represented as a read-only string).

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.49. dbi_result_bind_datetime

```
int dbi_result_bind_datetime(dbi_result Result, const char *fieldname, time_t *bindto)
```

Bind the specified variable to the specified field, which holds a DATE and/or TIME value.

Arguments

`Result`: The target query result.

`fieldname`: The name of the field to bind to.

`bindto`: A pointer to the variable that will be updated with the specified field's value.

Returns

0 upon success, -1 if there was an error

3.5.50. dbi_result_get_char_idx

```
signed char dbi_result_get_char_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the speficied field, which contains a character.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

3.5.51. dbi_result_get_uchar_idx

```
unsigned char dbi_result_get_uchar_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the speficied field, which contains an unsigned character.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

3.5.52. dbi_result_get_short_idx

```
short dbi_result_get_short_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the speficied field, which contains a short integer.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

3.5.53. dbi_result_get_ushort_idx

```
unsigned short dbi_result_get_ushort_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the speficied field, which contains an unsigned short integer.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

3.5.54. dbi_result_get_long_idx

```
long dbi_result_get_long_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the speficied field, which contains a long integer.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

3.5.55. dbi_result_get_ulong_idx

```
unsigned long dbi_result_get_ulong_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the speficied field, which contains an unsigned long integer.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

3.5.56. dbi_result_get_longlong_idx

```
long long dbi_result_get_longlong_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the speficied field, which contains a long long integer.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

3.5.57. dbi_result_get_ulonglong_idx

```
unsigned long long dbi_result_get_ulonglong_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the speficied field, which contains an unsigned long long integer.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

3.5.58. dbi_result_get_float_idx

```
float dbi_result_get_float_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the speficied field, which contains a floating-point number.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

3.5.59. dbi_result_get_double_idx

```
double dbi_result_get_double_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the speficied field, which contains a double-precision fractional number.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

3.5.60. dbi_result_get_string_idx

```
const char *dbi_result_get_string_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the speficied field, which contains a string. The string may not be modified, and may not necessairly persist between row fetches.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

3.5.61. dbi_result_get_binary_idx

```
const unsigned char *dbi_result_get_binary_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the speficied field, which contains binary BLOB data. The data may not be modified, and may not necessarily persist between row fetches.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

3.5.62. dbi_result_get_string_copy_idx

```
char *dbi_result_get_string_copy_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the speficied field, which contains a string. The newly allocated string may be modified by the host program, but the program is responsible for freeing the string.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

3.5.63. dbi_result_get_binary_copy_idx

```
unsigned char *dbi_result_get_binary_copy_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the speficied field, which contains binary BLOB data. The newly allocated memory may be modified by the host program, but the program is responsible for freeing the data.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

3.5.64. dbi_result_get_enum_idx

```
const char *dbi_result_get_enum_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the speficied field, which contains an ENUM (which will be represented as a read-only string).

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

3.5.65. dbi_result_get_set_idx

```
const char *dbi_result_get_set_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the speficied field, which contains a SET (which will be represented as a read-only string).

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

3.5.66. dbi_result_get_datetime_idx

```
time_t dbi_result_get_datetime_idx(dbi_result Result, unsigned int idx)
```

Fetch the data stored in the specified field, which contains a DATE and/or TIME value.

Arguments

`Result`: The target query result.

`idx`: The index of the target field (starting at 1).

Returns

The data stored in the specified field.

Appendix A. GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML

designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section’s title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of

all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and

conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.