MySQL++ Reference Manual

1.7

Generated by Doxygen 1.2.18

Sat Apr 30 07:30:04 2005

Contents

1 MySQL++ Reference Manual 1

2 MySQL++ Namespace Index 3

2.1 MySQL++ Namespace List 3

3 MySQL++ Hierarchical Index 5

3.1 MySQL++ Class Hierarchy 5

4 MySQL++ Compound Index 7

4.1 MySQL++ Compound List 7

5 MySQL++ File Index 9

5.1 MySQL++ File List 9

6 MySQL++ Namespace Documentation 11

6.1 mysqlpp Namespace Reference 11

7 MySQL++ Class Documentation 19

7.1 mysqlpp::BadConversion Class Reference 19

7.2 mysqlpp::BadFieldName Class Reference 20

7.3 mysqlpp::BadNullConversion Class Reference 21

7.4 mysqlpp::BadQuery Class Reference 22

7.5 mysqlpp::ColData_Tmpl< Str > Class Template Reference 23

7.6 mysqlpp::Connection Class Reference 24

7.7 mysqlpp::const_string Class Reference 28

7.8 mysqlpp::const_subscript_container< OnType, ValueType, ReturnType, SizeType, DiffType > Class Template Reference 29

7.9 mysqlpp::cstr_equal_to Struct Reference 30

7.10 mysqlpp::cstr_greater Struct Reference 31

7.11 mysqlpp::cstr_greater_equal Struct Reference 32

7.12 mysqlpp::cstr_less Struct Reference 33

7.13 mysqlpp::cstr_less_equal Struct Reference 34
7.14 mysqlpp::cstr_not_equal_to Struct Reference 35
7.15 mysqlpp::Date Struct Reference 36
7.16 mysqlpp::DateTime Struct Reference 37
7.17 mysqlpp::equal_list_b< Seq1, Seq2, Manip > Struct Template Reference 39
7.18 mysqlpp::equal_list_ba< Seq1, Seq2, Manip > Struct Template Reference 40
7.19 mysqlpp::escape_type1 Struct Reference 41
7.20 mysqlpp::escape_type2 Struct Reference 42
7.21 mysqlpp::FieldNames Class Reference 43
7.22 mysqlpp::Fields Class Reference 44
7.23 mysqlpp::FieldTypes Class Reference 45
7.24 mysqlpp::ignore_type2 Struct Reference 46
7.25 mysqlpp::mysql_date Struct Reference 47
7.26 mysqlpp::mysql_dt_base Struct Reference 49
7.27 mysqlpp::mysql_time Struct Reference 50
7.28 mysqlpp::mysql_type_info Class Reference 52
7.29 mysqlpp::MysqlCmp< BinaryPred, CmpType > Class Template Reference 54
7.30 mysqlpp::MysqlCmpCStr< BinaryPred > Class Template Reference 55
7.31 mysqlpp::Null< Type, Behavior > Class Template Reference 56
7.32 mysqlpp::null_type Class Reference 57
7.33 mysqlpp::NullisBlank Struct Reference 58
7.34 mysqlpp::NullisNull Struct Reference 59
7.35 mysqlpp::NullisZero Struct Reference 60
7.36 mysqlpp::Query Class Reference 61
7.37 mysqlpp::ResNSel Class Reference 66
7.38 mysqlpp::Result Class Reference 67
7.39 mysqlpp::ResUse Class Reference 69
7.40 mysqlpp::Row Class Reference 72
7.41 mysqlpp::RowTemplate< ThisType, Res > Class Template Reference 74
7.42 mysqlpp::Set< Container > Class Template Reference 75
7.43 mysqlpp::simp_list_b< Iter > Class Template Reference 76
7.44 mysqlpp::SQLParseElement Struct Reference 77
7.45 mysqlpp::SQLQuery Class Reference 78
7.46 mysqlpp::SQLQueryNEParms Class Reference 80
7.47 mysqlpp::SQLQueryParms Class Reference 81
7.48 mysqlpp::SQLString Class Reference 83

7.49 mysqlpp::subscript_iterator< OnType, ReturnType, SizeType, DiffType > Class Template Reference 84
7.50 mysqlpp::Time Struct Reference 85
7.51 mysqlpp::tiny_int Class Reference 86
7.52 mysqlpp::type_info_cmp Struct Reference 87
7.53 mysqlpp::value_list_b< Seq, Manip > Struct Template Reference 88
7.54 mysqlpp::value_list_ba< Seq, Manip > Struct Template Reference 89

8 MySQL++ File Documentation 91

8.1 coldata.h File Reference 91
8.2 compare.h File Reference 94
8.3 connection.h File Reference 95
8.4 const_string.h File Reference 97
8.5 convert.h File Reference 99
8.6 datetime.h File Reference 101
8.7 defs.h File Reference 103
8.8 exceptions.h File Reference 105
8.9 field_names.h File Reference 107
8.10 field_types.h File Reference 108
8.11 fields.h File Reference 109
8.12 manip.h File Reference 110
8.13 myset.h File Reference 112
8.14 mysql++.h File Reference 114
8.15 mysql++.hh File Reference 116
8.16 null.h File Reference 117
8.17 platform.h File Reference 119
8.18 query.h File Reference 120
8.19 resiter.h File Reference 122
8.20 result.h File Reference 123
8.21 row.h File Reference 125
8.22 sql_query.h File Reference 126
8.23 sql_string.h File Reference 128
8.24 sqlplus.hh File Reference 130
8.25 stream2string.h File Reference 131
8.26 string_util.h File Reference 132
8.27 tiny_int.h File Reference 134
8.28 type_info.h File Reference 135

8.29 vallist.h File Reference 137

Chapter 1

MySQL++ Reference Manual

1.0.1 Getting Started

Currently, there is no tutorial or other new user documentation. (There is an old manual in the doc/old directory, but it's badly outdated.)

Until we can make some good user documentation, the best way to get started is to read through the example programs. Start with the simple1 program, and progress from there. The custom∗ programs are of particular interest. See the examples/README file for further details.

1.0.2 Major Classes

In MySQL++, the main user-facing classes are **mysqlpp::Connection** (p. 24), **mysqlpp::Query** (p. 61), **mysqlpp::Result** (p. 67), and **mysqlpp::Row** (p. 72).

In addition, MySQL++ has a mechanism called Specialized SQL Structures (SSQLS), which allow you to create C++ structures that parallel the definition of the tables in your database schema. These let you manipulate the data in your database using native C++ data structures. Many programs using this feature never directly use SQL, because MySQL++ generates it all automatically. See the custom.∗ programs in the examples directory to see how to use this feature. It is not necessary to use MySQL++ this way, but it sure makes some things a lot easier.

1.0.3 Major Files

The only two header files your program ever needs to include are **mysql++.h**, and optionally custom.h. (The latter implements the SSQLS mechanism.) All of the other files are used within the library only.

By the way, if, when installing this package, you didn't put the headers into their own subdirectory, you might consider reinstalling the package to remedy that. MySQL++ has a number of generically-named files (**convert.h,** (p. 99) **fields.h,** (p. 109) **row.h**...), so it's best to put them into a separate directory where they can't interfere with other code on your system. If you're on a Unixy platform, you do this by passing the `--includedir` option to the `configure` script. See the package's main README file for details.

1.0.4 A Brief History of MySQL++

MySQL++ was created in 1998 by Kevin Atkinson. It started out MySQL-specific, but there were early efforts to try and make it database-independent, and call it SQL++. This is where the old library name "sqlplus" came from. This is also why the old versions prefixed some class names with "Mysql" but not others: the others were supposed to be the database-independent parts.

Then in 1999, Sinisa Miliojevic unofficially took over maintenance of the library, releasing versions 1.0 and 1.1. (All of Kevin's releases were pre-1.0 point releases.) Kevin gave over maintenance to Sinisa officially with 1.2, and Sinisa went on to maintain the library through 1.7.9, released in mid-2001. Since Sinisa is an employee of `MySQL AB`, it seems to be during this time that the dream of multiple-database compatibility died.

With version 1.7.9, MySQL++ went into a period of stasis, lasting over three years. During this time, Sinisa ran the MySQL++ mailing list and supported its users, but made no new releases. There were many patches submitted during this period, some of which were ignored, others which were just put on the MySQL++ web site for people to try. A lot of these patches were mutually-incompatible, and not all of them gave a fully-functional copy of MySQL++. Most of them centered on GCC compatibility, because GCC 3 was created around this same time. The continual tightening of the rules starting with GCC 3.0 exposed a lot of places in MySQL++ that used deprecated and non-conforming facilities.

In early August of 2004, the current maintainer (Warren Young) got fed up with this situation and took over. He released 1.7.10 later that month.

This Doxygen-based reference manual replaces section 5 from the old v1.7.9 manual. The remaining sections of that manual are in the new user manual, in ../doc/userman.

1.0.5 You Have Questions...

If you want to email someone to ask questions about this library, we greatly prefer that you send mail to the MySQL++ mailing list, which you can subscribe to here: `http://lists.mysql.com/plusplus`

That mailing list is archived, so if you have questions, do a search to see if the question has been asked before.

You may find people's individual email addresses in various files within the MySQL++ distribution. Please do not send mail to them unless you are sending something that is inherently personal. Questions that are about MySQL++ usage may well be ignored if you send them to our personal email accounts. Those of us still active in MySQL++ development monitor the mailing list, so you aren't getting any extra "coverage" by sending messages to those addresses in addition to the mailing list.

Chapter 2

MySQL++ Namespace Index

2.1 MySQL++ Namespace List

Here is a list of all documented namespaces with brief descriptions:

mysqlpp 11

Chapter 3

MySQL++ Hierarchical Index

3.1 MySQL++ Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- mysqlpp::BadConversion 19
- mysqlpp::BadFieldName 20
- mysqlpp::BadNullConversion 21
- mysqlpp::BadQuery 22
- mysqlpp::ColData_Tmpl< Str > 23
- mysqlpp::Connection 24
- mysqlpp::const_string 28
- mysqlpp::const_subscript_container< OnType, ValueType, ReturnType, SizeType, DiffType > 29
- mysqlpp::const_subscript_container< Fields, Field > 29
  - mysqlpp::Fields 44
- mysqlpp::const_subscript_container< Result, Row, const Row > 29
  - mysqlpp::Result 67
- mysqlpp::const_subscript_container< Row, ColData, const ColData > 29
  - mysqlpp::Row 72
- mysqlpp::cstr_equal_to 30
- mysqlpp::cstr_greater 31
- mysqlpp::cstr_greater_equal 32
- mysqlpp::cstr_less 33
- mysqlpp::cstr_less_equal 34
- mysqlpp::cstr_not_equal_to 35
- mysqlpp::equal_list_b< Seq1, Seq2, Manip > 39
- mysqlpp::equal_list_ba< Seq1, Seq2, Manip > 40
- mysqlpp::escape_type1 41
- mysqlpp::escape_type2 42
- mysqlpp::FieldNames 43
- mysqlpp::FieldTypes 45
- mysqlpp::ignore_type2 46
- mysqlpp::mysql_dt_base 49
  - mysqlpp::mysql_date 47
    - mysqlpp::Date 36
    - mysqlpp::DateTime 37

  - mysqlpp::mysql_time 50
    - mysqlpp::DateTime 37
    - mysqlpp::Time 85
- mysqlpp::mysql_type_info 52
- mysqlpp::Null< Type, Behavior > 56
- mysqlpp::null_type 57
- mysqlpp::NullisBlank 58
- mysqlpp::NullisNull 59
- mysqlpp::NullisZero 60
- mysqlpp::ResNSel 66
- mysqlpp::ResUse 69
  - mysqlpp::Result 67
- mysqlpp::RowTemplate< ThisType, Res > 74
- mysqlpp::RowTemplate< Row, ResUse > 74
  - mysqlpp::Row 72
- mysqlpp::Set< Container > 75
- mysqlpp::simp_list_b< Iter > 76
- mysqlpp::SQLParseElement 77
- mysqlpp::SQLQuery 78
  - mysqlpp::Query 61
- mysqlpp::SQLQueryNEParms 80
- mysqlpp::SQLQueryParms 81
- mysqlpp::SQLString 83
- mysqlpp::subscript_iterator< OnType, ReturnType, SizeType, DiffType > 84
- mysqlpp::tiny_int 86
- mysqlpp::type_info_cmp 87
- unary_function
  - mysqlpp::MysqlCmp< BinaryPred, CmpType > 54
  - mysqlpp::MysqlCmp< BinaryPred, const char ∗ > 54
    - mysqlpp::MysqlCmpCStr< BinaryPred > 55
- mysqlpp::value_list_b< Seq, Manip > 88
- mysqlpp::value_list_ba< Seq, Manip > 89

Chapter 4

MySQL++ Compound Index

4.1 MySQL++ Compound List

Here are the classes, structs, unions and interfaces with brief descriptions:

mysqlpp::BadConversion (Exception thrown when a bad conversion takes place) . . 19
mysqlpp::BadFieldName (Exception thrown when a requested named field doesn't exist) 20
mysqlpp::BadNullConversion (Exception thrown when you attempt to convert a SQL null to an incompatible type) 21
mysqlpp::BadQuery (Exception thrown when MySQL encounters a problem while processing your query) 22
mysqlpp::ColData_Tmpl< Str > (Base class for auto-converting column data. Do not use directly) 23
mysqlpp::Connection (Manages the connection to the MySQL database) 24
mysqlpp::const_string (Wrapper for `const char*` to make it behave in a way more useful to MySQL++) 28
mysqlpp::const_subscript_container< OnType, ValueType, ReturnType, SizeType, DiffType > (A base class that one derives from to become a random access container, which can be accessed with subscript notation) 29
mysqlpp::cstr_equal_to (Documentation needed!) 30
mysqlpp::cstr_greater (Documentation needed!) 31
mysqlpp::cstr_greater_equal (Documentation needed!) 32
mysqlpp::cstr_less (Documentation needed!) 33
mysqlpp::cstr_less_equal (Documentation needed!) 34
mysqlpp::cstr_not_equal_to (Documentation needed!) 35
mysqlpp::Date (Holds MySQL dates) 36
mysqlpp::DateTime (A combination of the **Date** (p. 36) and **Time** (p. 85) classes for holding MySQL DateTimes) 37
mysqlpp::equal_list_b< Seq1, Seq2, Manip > (Documentation needed!) 39
mysqlpp::equal_list_ba< Seq1, Seq2, Manip > (Documentation needed!) 40
mysqlpp::escape_type1 (Documentation needed!) 41
mysqlpp::escape_type2 (Documentation needed!) 42
mysqlpp::FieldNames (Holds a list of SQL field names) 43
mysqlpp::Fields (A container similar to `std::vector` for holding mysqlpp::Field records) 44
mysqlpp::FieldTypes (A vector of SQL field types) 45
mysqlpp::ignore_type2 (Documentation needed!) 46
mysqlpp::mysql_date (Base class of **Date** (p. 36)) 47

mysqlpp::mysql_dt_base (Base class for **mysql_date** (p. 47) and **mysql_time** (p. 50)) 49
mysqlpp::mysql_time (Base class of **Time** (p. 85)) 50
mysqlpp::mysql_type_info (Holds basic type information for ColData) 52
mysqlpp::MysqlCmp< BinaryPred, CmpType > (Documentation needed!) 54
mysqlpp::MysqlCmpCStr< BinaryPred > (Documentation needed!) 55
mysqlpp::Null< Type, Behavior > (Container class for holding SQL nulls) 56
mysqlpp::null_type (The type of the global mysqlpp::null object) 57
mysqlpp::NullisBlank (Used for the behavior parameter for template **Null** (p. 56)) . 58
mysqlpp::NullisNull (Used for the behavior parameter for template **Null** (p. 56)) . . 59
mysqlpp::NullisZero (Used for the behavior parameter for template **Null** (p. 56)) . . 60
mysqlpp::Query (A class for building and executing SQL queries) 61
mysqlpp::ResNSel (Holds the information on the success of queries that don't return any results) 66
mysqlpp::Result (This class manages SQL result sets) 67
mysqlpp::ResUse (A basic result set class, for use with "use" queries) 69
mysqlpp::Row (Manages rows from a result set) 72
mysqlpp::RowTemplate< ThisType, Res > (Documentation needed!) 74
mysqlpp::Set< Container > (A special std::set derivative for holding MySQL data sets) 75
mysqlpp::simp_list_b< Iter > (Documentation needed!) 76
mysqlpp::SQLParseElement (Documentation needed!) 77
mysqlpp::SQLQuery (The base class for **mysqlpp::Query** (p. 61)) 78
mysqlpp::SQLQueryNEParms (Exception thrown when not enough parameters are provided) 80
mysqlpp::SQLQueryParms (This class holds the parameter values for filling template queries) 81
mysqlpp::SQLString (A specialized `std::string` that will convert from any valid MySQL type) 83
mysqlpp::subscript_iterator< OnType, ReturnType, SizeType, DiffType > (Iterator that can be subscripted) 84
mysqlpp::Time (Holds MySQL times) 85
mysqlpp::tiny_int (Class for holding an SQL `tiny_int` object) 86
mysqlpp::type_info_cmp (Documentation needed!) 87
mysqlpp::value_list_b< Seq, Manip > (Documentation needed!) 88
mysqlpp::value_list_ba< Seq, Manip > (Documentation needed!) 89

Chapter 5

MySQL++ File Index

5.1 MySQL++ File List

Here is a list of all documented files with brief descriptions:

coldata.h (Declares classes for converting string data to any of the basic C types) . . . 91
compare.h 94
connection.h (Declares the Connection class) 95
const_string.h (Declares a wrapper for `const char*` which behaves in a way more useful to MySQL++) 97
convert.h (Declares various string-to-integer type conversion templates) 99
datetime.h (Declares classes to add MySQL-compatible date and time types to C++'s type system) 101
defs.h (Standard definitions used all across the library, particularly things that don't fit well anywhere else) 103
exceptions.h (Declares the MySQL++-specific exception classes) 105
field_names.h (Declares a class to hold a list of field names) 107
field_types.h (Declares a class to hold a list of SQL field type info) 108
fields.h (Declares a class for holding information about a set of fields) 109
manip.h (Declares `std::ostream` manipulators useful with SQL syntax) 110
myset.h (Declares templates for generating custom containers used elsewhere in the library) 112
mysql++.h (The main MySQL++ header file. It simply includes all of the other header files (except for custom.h) in the proper order) 114
mysql++.hh (Deprecated backwards-compatibility header. Use **mysql++.h** in new code instead) 116
null.h (Declares classes that implement SQL "null" semantics within C++'s type system) 117
platform.h (This file includes things that help the rest of MySQL++) 119
query.h (Defines the user-facing Query class, which is used to build up SQL queries, and execute them) 120
resiter.h (Declares templates for adapting existing classes to be iteratable random-access containers) 122
result.h (Declares classes for holding SQL query result sets) 123
row.h (Declares the classes for holding row data from a result set) 125
sql_query.h (Declares the base class for **mysqlpp::Query** (p. 61), plus some utility classes to be used with it) 126
sql_string.h (Declares an `std::string` derivative that adds some things needed within the library) 128

sqlplus.hh (Deprecated backwards-compatibility header. Use **mysql++.h** in new code instead) 130
stream2string.h (Declares an adapter that converts something that can be inserted into a C++ stream into a string type) 131
string_util.h (Declares string-handling utility functions used within the library) 132
tiny_int.h (Declares class for holding a SQL tiny_int) 134
type_info.h (Declares classes that provide an interface between the SQL and C++ type systems) 135
vallist.h (Declares templates for holding lists of values) 137

Chapter 6

MySQL++ Namespace Documentation

6.1 mysqlpp Namespace Reference

Compounds

- class **BadConversion**

  Exception thrown when a bad conversion takes place.

- class **BadFieldName**

  Exception thrown when a requested named field doesn't exist.

- class **BadNullConversion**

  Exception thrown when you attempt to convert a SQL null to an incompatible type.

- class **BadQuery**

  Exception thrown when MySQL encounters a problem while processing your query.

- class **ColData_Tmpl**

  Base class for auto-converting column data. Do not use directly.

- class **Connection**

  Manages the connection to the MySQL database.

- class **const_string**

  Wrapper for `const char*` *to make it behave in a way more useful to MySQL++.*

- class **const_subscript_container**

  A base class that one derives from to become a random access container, which can be accessed with subscript notation.

- struct **cstr_equal_to**

  Documentation needed!

- struct **cstr_greater**

  Documentation needed!

- struct **cstr_greater_equal**

  Documentation needed!

- struct **cstr_less**

  Documentation needed!

- struct **cstr_less_equal**

  Documentation needed!

- struct **cstr_not_equal_to**

  Documentation needed!

- struct **Date**

  Holds MySQL dates.

- struct **DateTime**

  A combination of the **Date** (p. 36) *and* **Time** (p. 85) *classes for holding MySQL DateTimes.*

- struct **equal_list_b**

  Documentation needed!

- struct **equal_list_ba**

  Documentation needed!

- struct **escape_type1**

  Documentation needed!

- struct **escape_type2**

  Documentation needed!

- class **FieldNames**

  Holds a list of SQL field names.

- class **Fields**

  A container similar to `std::vector` *for holding mysqlpp::Field records.*

- class **FieldTypes**

  A vector of SQL field types.

- struct **ignore_type2**

  Documentation needed!

- struct **mysql_date**

  Base class of **Date** (p. 36).

- struct **mysql_dt_base**

  Base class for **mysql_date** (p. 47) *and* **mysql_time** (p. 50).

- struct **mysql_time**

  Base class of **Time** (p. 85).

- class **mysql_type_info**

  Holds basic type information for ColData.

- class **MysqlCmp**

  Documentation needed!

- class **MysqlCmpCStr**

  Documentation needed!

- class **Null**

  Container class for holding SQL nulls.

- class **null_type**

  The type of the global mysqlpp::null object.

- struct **NullisBlank**

  Used for the behavior parameter for template **Null** (p. 56).

- struct **NullisNull**

  Used for the behavior parameter for template **Null** (p. 56).

- struct **NullisZero**

  Used for the behavior parameter for template **Null** (p. 56).

- class **Query**

  A class for building and executing SQL queries.

- class **ResNSel**

  Holds the information on the success of queries that don't return any results.

- class **Result**

  This class manages SQL result sets.

- class **ResUse**

  A basic result set class, for use with "use" queries.

- class **Row**

  Manages rows from a result set.

- class **RowTemplate**

  Documentation needed!

- class **Set**

  A special std::set derivative for holding MySQL data sets.

- class **simp_list_b**

Documentation needed!

- struct **SQLParseElement**

  Documentation needed!

- class **SQLQuery**

  The base class for **mysqlpp::Query** (p. 61).

- class **SQLQueryNEParms**

  Exception thrown when not enough parameters are provided.

- class **SQLQueryParms**

  This class holds the parameter values for filling template queries.

- class **SQLString**

  A specialized `std::string` *that will convert from any valid MySQL type.*

- class **subscript_iterator**

  Iterator that can be subscripted.

- struct **Time**

  Holds MySQL times.

- class **tiny_int**

  Class for holding an SQL `tiny_int` *object.*

- struct **type_info_cmp**

  Documentation needed!

- struct **value_list_b**

  Documentation needed!

- struct **value_list_ba**

  Documentation needed!

Typedefs

- typedef **ColData_Tmpl**< **const_string** > **ColData**

  The type that is returned by constant rows.

- typedef **ColData_Tmpl**< std::string > **MutableColData**

  The type that is returned by mutable rows.

Enumerations

- enum **quote_type0** { **quote** }
- enum **quote_only_type0** { **quote_only** }
- enum **quote_double_only_type0** { **quote_double_only** }
- enum **escape_type0**
- enum **do_nothing_type0** { **do_nothing** }
- enum **ignore_type0** { **ignore** }
- enum **query_reset**

  Used for indicating whether a query object should auto-reset or not.

Functions

- template<class BinaryPred, class CmpType> **MysqlCmp**< BinaryPred, CmpType > **mysql_cmp** (uint i, const BinaryPred &func, const CmpType &cmp2)

  For comparing any two objects, as long as they can be converted to **SQLString** (p. 83)*.*

- template<class BinaryPred> **MysqlCmpCStr**< BinaryPred > **mysql_cmp_cstr** (uint i, const BinaryPred &func, const char ∗cmp2)

  For comparing anything to a `const char*`*.*

- template<class Type, class Behavior> std::ostream & **operator**<< (std::ostream &o, const **Null**< Type, Behavior > &n)
- template<class Strng, class T> Strng **stream2string** (const T &object)

  Converts a stream-able object to any type that can be initialized from an `std::string`*.*

- void **strip** (std::string &s)

  Strips blanks at left and right ends.

- void **escape_string** (std::string &s)

  C++ equivalent of mysql_escape_string().

- void **str_to_upr** (std::string &s)

  Changes case of string to upper.

- void **str_to_lwr** (std::string &s)

  Changes case of string to lower.

- void **strip_all_blanks** (std::string &s)

  Removes all blanks.

- void **strip_all_non_num** (std::string &s)

  Removes all non-numerics.

6.1.1 Detailed Description

All global symbols in MySQL++ are in namespace mysqlpp. This is needed because many symbols are rather generic (e.g. **Row** (p. 72), **Query** (p. 61)...), so there is a serious danger of conflicts.

6.1.2 Enumeration Type Documentation

6.1.2.1 enum mysqlpp::do_nothing_type0

The 'do_nothing' manipulator.

Does exactly what it says: nothing. Used as a dummy manipulator when you are required to use some manipulator but don't want anything to be done to the following item. When used with **SQLQueryParms** (p. 81) it will make sure that it does not get formatted in any way, overriding any setting set by the template query.

Enumeration values:

do_nothing	insert into a std::ostream to override manipulation of next item

6.1.2.2 enum mysqlpp::escape_type0

The 'escape' manipulator.

Calls mysql_escape_string() in the MySQL C API on the following argument to prevent any special SQL characters from being interpreted.

6.1.2.3 enum mysqlpp::ignore_type0

The 'ignore' manipulator.

Only valid when used with **SQLQueryParms** (p. 81). It's a dummy manipulator like the `do_nothing manipulator`, except that it will not override formatting set by the template query. It is simply ignored.

Enumeration values:

ignore	insert into a std::ostream as a dummy manipulator

6.1.2.4 enum mysqlpp::quote_double_only_type0

The 'double_quote_only' manipulator.

Similar to `quote_only manipulator`, except that it uses double quotes instead of single quotes.

Enumeration values:

quote_double_only	insert into a std::ostream to double-quote next item

6.1.2.5 enum mysqlpp::quote_only_type0

The 'quote_only' manipulator.

Similar to `quote manipulator`, except that it doesn't escape special SQL characters.

Enumeration values:

quote_only	insert into a std::ostream to single-quote next item

6.1.2.6 enum mysqlpp::quote_type0

The standard 'quote' manipulator.

Insert this into a stream to put single quotes around the next item in the stream, and escape characters within it that are 'special' in SQL. This is the most generally useful of the manipulators.

Enumeration values:

quote insert into a std::ostream to single-quote and escape next item

6.1.3 Function Documentation

6.1.3.1 template<class BinaryPred, class CmpType> MysqlCmp<BinaryPred, CmpType> mysql_cmp (uint *i*, const BinaryPred & *func*, const CmpType & *cmp2*)

For comparing any two objects, as long as they can be converted to **SQLString** (p. 83).

This template is for creating predicate objects for use with STL algorithms like std::find_if().

This is a more generic form of **mysql_cmp_cstr()** (p. 17) which will work with any C++ type that can be converted to **mysqlpp::SQLString** (p. 83). This is not nearly as efficient as that function, so use this only when absolutely necessary.

Parameters:

i field index number

func one of the functors in **compare.h**, or any compatible functor

cmp2 what to compare to

6.1.3.2 template<class BinaryPred> MysqlCmpCStr<BinaryPred> mysql_cmp_cstr (uint *i*, const BinaryPred & *func*, const char ∗ *cmp2*)

For comparing anything to a `const char*`.

This template is for creating predicate objects for use with STL algorithms like std::find_if().

Parameters:

i field index number

func one of `cstr_equal_to()`, `cstr_not_equal_to()`, `cstr_less()`, `cstr_less_equal()`, `cstr_less_equal()`, or `cstr_less_equal()`.

cmp2 what to compare to

6.1.3.3 template<class Type, class Behavior> std::ostream& operator<< (std::ostream & *o*, const Null< Type, Behavior > & *n*) `[inline]`

6.1.3.4 template<class Strng, class T> Strng stream2string (const T & *object*)

Converts a stream-able object to any type that can be initialized from an `std::string`.

This adapter takes any object that has an `out_stream()` member function and converts it to a string type. An example of such a type within the library is **mysqlpp::Date** (p. 36).

Chapter 7

MySQL++ Class Documentation

7.1 mysqlpp::BadConversion Class Reference

Exception thrown when a bad conversion takes place.

`#include <exceptions.h>`

7.1.1 Detailed Description

Exception thrown when a bad conversion takes place.

The documentation for this class was generated from the following file:

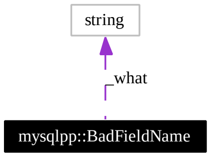
- **exceptions.h**

7.2 mysqlpp::BadFieldName Class Reference

Exception thrown when a requested named field doesn't exist.

`#include <exceptions.h>`

Collaboration diagram for mysqlpp::BadFieldName:



7.2.1 Detailed Description

Exception thrown when a requested named field doesn't exist.

Thrown by Row::lookup_by_name() when you pass a field name that isn't in the result set.

The documentation for this class was generated from the following file:

- **exceptions.h**

7.3 mysqlpp::BadNullConversion Class Reference

Exception thrown when you attempt to convert a SQL null to an incompatible type.

`#include <exceptions.h>`

7.3.1 Detailed Description

Exception thrown when you attempt to convert a SQL null to an incompatible type.

The documentation for this class was generated from the following file:

- **exceptions.h**

7.4 mysqlpp::BadQuery Class Reference

Exception thrown when MySQL encounters a problem while processing your query.

`#include <exceptions.h>`

Public Attributes

- const std::string **error**

  contains explanation why query was bad

7.4.1 Detailed Description

Exception thrown when MySQL encounters a problem while processing your query.

This is the most generic MySQL++ exception. It is thrown when your SQL syntax is incorrect, or a field you requested doesn't exist in the database, or....

The documentation for this class was generated from the following file:

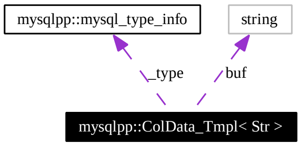
- **exceptions.h**

7.5 mysqlpp::ColData_Tmpl< Str > Class Template Reference

Base class for auto-converting column data. Do not use directly.

`#include <coldata.h>`

Collaboration diagram for mysqlpp::ColData_Tmpl< Str >:



Public Methods

- **mysql_type_info** **type** ()

  Get this object's current MySQL type.

- bool **quote_q** () const

  Returns true if data of this type should be quoted, false otherwise.

- bool **escape_q** () const

  Returns true if data of this type should be escaped, false otherwise.

7.5.1 Detailed Description

template<class Str> class mysqlpp::ColData_Tmpl< Str >

Base class for auto-converting column data. Do not use directly.

A smart string. It will automatically convert itself to any of the basic C types.

When used with binary operators it will automatically convert itself to the type used on the other side of the operator if it is a basic type. However, be careful when using it like this:

```
ColData("12.86") + 2
```

will return 14 because 2 is an integer. To get the result you wanted, you must say:

```
ColData("12.86") + 2.0
```

If this type of thing scares you, define the micro NO_BINARY_OPERS to disable this behavior.

This class also has some basic information about the type of data stored in it.

Do not use this class directly. Use the typedef ColData or MutableColData instead.

The documentation for this class was generated from the following file:

- **coldata.h**

7.6 mysqlpp::Connection Class Reference

Manages the connection to the MySQL database.

`#include <connection.h>`

Public Methods

- **Connection** ()

  Create object without connecting it to the MySQL server.

- **Connection** (bool te)
- **Connection** (const char *db, const char *host="", const char *user="", const char *passwd="", bool te=true)

  For connecting to database without any special options.

- **Connection** (const char *db, const char *host, const char *user, const char *passwd, uint port, my_bool compress=0, unsigned int connect_timeout=60, bool te=true, cchar *socket_name=0, unsigned int client_flag=0)

  Connect to database, allowing you to specify all connection parameters.

- bool **connect** (cchar *db="", cchar *host="", cchar *user="", cchar *passwd="")

  Open connection to MySQL database.

- bool **real_connect** (cchar *db="", cchar *host="", cchar *user="", cchar *passwd="", uint port=0, my_bool compress=0, unsigned int connect_timeout=60, cchar *socket_name=0, unsigned int client_flag=0)

  Connect to database after object is created.

- void **close** ()

  Close connection to MySQL server.

- std::string **info** ()

  Calls MySQL C API function `mysql_info()` *and returns result as a C++ string.*

- bool **connected** () const

  return true if connection was established successfully

- bool **success** () const

  Return true if the last query was successful.

- **Query query** ()

  Return a new query object.

- **operator bool** ()

  Alias for **success()** (p. 27).

- const char * **error** ()

  Return last error message.

7.6.1 Detailed Description

Manages the connection to the MySQL database.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 mysqlpp::Connection::Connection ()

Create object without connecting it to the MySQL server.

Use **real_connect()** (p. 27) method to establish the connection.

7.6.2.2 mysqlpp::Connection::Connection (bool *te*)

Same as **default ctor** (p. 25) except that it allows you to choose whether exceptions are enabled.

Parameters:

te if true, exceptions are thrown on errors

7.6.2.3 mysqlpp::Connection::Connection (const char ∗ *db*, const char ∗ *host* = "", const char ∗ *user* = "", const char ∗ *passwd* = "", bool *te* = true)

For connecting to database without any special options.

This constructor takes the minimum parameters needed for most programs' use of MySQL. There is a `more complicated` constructor that lets you specify everything that the C API function `mysql_real_connect()` does.

Parameters:

db name of database to use

host host name or IP address of MySQL server, or 0 if server is running on the same host as your program

user user name to log in under, or 0 to use the user name this program is running under

passwd password to use when logging in

te if true, throw exceptions on errors

7.6.2.4 mysqlpp::Connection::Connection (const char ∗ *db*, const char ∗ *host*, const char ∗ *user*, const char ∗ *passwd*, uint *port*, my_bool *compress* = 0, unsigned int *connect_timeout* = 60, bool *te* = true, cchar ∗ *socket_name* = 0, unsigned int *client_flag* = 0)

Connect to database, allowing you to specify all connection parameters.

This constructor allows you to most fully specify the options used when connecting to the MySQL database. It is the thinnest layer in MySQL++ over the MySQL C API function `mysql_real_connect()`.

Parameters:

db name of database to use

host host name or IP address of MySQL server, or 0 if server is running on the same host as your program

user user name to log in under, or 0 to use the user name this program is running under

passwd password to use when logging in

port TCP port number MySQL server is listening on, or 0 to use default value

compress if true, compress data passing through connection, to save bandwidth at the expense of CPU time

connect_timeout max seconds to wait for server to respond to our connection attempt

te if true, throw exceptions on errors

socket_name Unix domain socket server is using, if connecting to MySQL server on the same host as this program running on, or 0 to use default name

client_flag special connection flags. See MySQL C API documentation for `mysql_real_connect()` for details.

7.6.3 Member Function Documentation

7.6.3.1 void mysqlpp::Connection::close () `[inline]`

Close connection to MySQL server.

Closes the connection to the MySQL server.

7.6.3.2 bool mysqlpp::Connection::connect (cchar ∗ *db* = "", cchar ∗ *host* = "", cchar ∗ *user* = "", cchar ∗ *passwd* = "")

Open connection to MySQL database.

Open connection to the MySQL server, using defaults for all but the most common parameters. It's better to use one of the connect-on-create constructors if you can.

See `this` for parameter documentation.

7.6.3.3 bool mysqlpp::Connection::connected () const `[inline]`

return true if connection was established successfully

Returns:
true if connection was established successfully

7.6.3.4 const char∗ mysqlpp::Connection::error () `[inline]`

Return last error message.

Simply wraps `mysql_error()` in the C API.

7.6.3.5 mysqlpp::Connection::operator bool () `[inline]`

Alias for **success()** (p. 27).

Alias for **success()** (p. 27) member function. Allows you to have code constructs like this:

```
Connection conn;
.... use conn
if (conn) {
    ... last SQL query was successful
}
else {
    ... error occurred in SQL query
}
```

7.6.3.6 Query mysqlpp::Connection::query ()

Return a new query object.

The returned query object is tied to this MySQL connection, so when you call a method like **execute()** (p. 63) on that object, the query is sent to the server this object is connected to.

7.6.3.7 bool mysqlpp::Connection::real_connect (cchar ∗ *db* = "", cchar ∗ *host* = "", cchar ∗ *user* = "", cchar ∗ *passwd* = "", uint *port* = 0, my_bool *compress* = 0, unsigned int *connect_timeout* = 60, cchar ∗ *socket_name* = 0, unsigned int *client_flag* = 0)

Connect to database after object is created.

It's better to use one of the connect-on-create constructors if you can.

Despite the name, this function is not a direct wrapper for the MySQL C API function `mysql_real_connect()`. It also sets some connection-related options using `mysql_options()`.

See `this` for parameter documentation.

7.6.3.8 bool mysqlpp::Connection::success () const `[inline]`

Return true if the last query was successful.

Return true if the most recent query was successful

The documentation for this class was generated from the following files:

- **connection.h**
- connection.cpp

7.7 mysqlpp::const_string Class Reference

Wrapper for `const char*` to make it behave in a way more useful to MySQL++.

`#include <const_string.h>`

Public Types

- typedef const_iterator **iterator**

  Same as const_iterator because the data cannot be changed.

7.7.1 Detailed Description

Wrapper for `const char*` to make it behave in a way more useful to MySQL++.

This class implements a small subset of the standard string class.

Objects are created from an existing `const char*` variable by copying the pointer only. Therefore, the object pointed to by that pointer needs to exist for at least as long as the **const_string** (p. 28) object that wraps it.

The documentation for this class was generated from the following file:

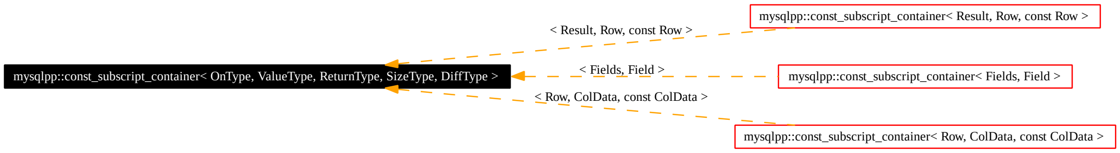
- **const_string.h**

7.8 mysqlpp::const_subscript_container< OnType, ValueType, ReturnType, SizeType, DiffType > Class Template Reference

A base class that one derives from to become a random access container, which can be accessed with subscript notation.

`#include <resiter.h>`

Inheritance diagram for mysqlpp::const_subscript_container< OnType, ValueType, ReturnType, SizeType, DiffType >:


mysqlpp::const_subscript_container< OnType, ValueType, ReturnType, SizeType, DiffType >
< Result, Row, const Row >
mysqlpp::const_subscript_container< Result, Row, const Row >
< Fields, Field >
mysqlpp::const_subscript_container< Fields, Field >
< Row, ColData, const ColData >
mysqlpp::const_subscript_container< Row, ColData, const ColData >

7.8.1 Detailed Description

template<class OnType, class ValueType, class ReturnType = const ValueType&, class SizeType = unsigned int, class DiffType = int> class mysqlpp::const_subscript_container< OnType, ValueType, ReturnType, SizeType, DiffType >

A base class that one derives from to become a random access container, which can be accessed with subscript notation.

OnType must have the member functions `operator[](SizeType)` and

The documentation for this class was generated from the following file:

- **resiter.h**

7.9 mysqlpp::cstr_equal_to Struct Reference

Documentation needed!

`#include <compare.h>`

7.9.1 Detailed Description

Documentation needed!

Document me!

The documentation for this struct was generated from the following file:

- **compare.h**

7.10 mysqlpp::cstr_greater Struct Reference

Documentation needed!

`#include <compare.h>`

7.10.1 Detailed Description

Documentation needed!

Document me!

The documentation for this struct was generated from the following file:

- **compare.h**

7.11 mysqlpp::cstr_greater_equal Struct Reference

Documentation needed!

`#include <compare.h>`

7.11.1 Detailed Description

Documentation needed!

Document me!

The documentation for this struct was generated from the following file:

- **compare.h**

7.12 mysqlpp::cstr_less Struct Reference

Documentation needed!

`#include <compare.h>`

7.12.1 Detailed Description

Documentation needed!

Document me!

The documentation for this struct was generated from the following file:

- **compare.h**

7.13 mysqlpp::cstr_less_equal Struct Reference

Documentation needed!

`#include <compare.h>`

7.13.1 Detailed Description

Documentation needed!

Document me!

The documentation for this struct was generated from the following file:

- **compare.h**

7.14 mysqlpp::cstr_not_equal_to Struct Reference

Documentation needed!

`#include <compare.h>`

7.14.1 Detailed Description

Documentation needed!

Document me!

The documentation for this struct was generated from the following file:

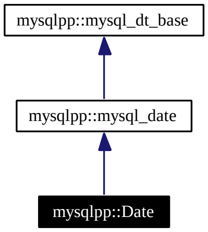
- **compare.h**

7.15 mysqlpp::Date Struct Reference

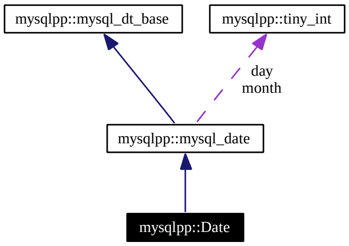
Holds MySQL dates.

`#include <datetime.h>`

Inheritance diagram for mysqlpp::Date:



Collaboration diagram for mysqlpp::Date:



7.15.1 Detailed Description

Holds MySQL dates.

Objects of this class can be inserted into streams, and initialized from a stream.

The documentation for this struct was generated from the following file:

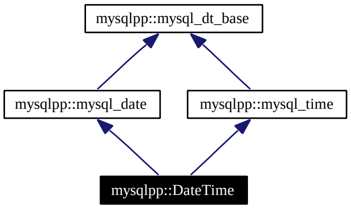
- **datetime.h**

7.16 mysqlpp::DateTime Struct Reference

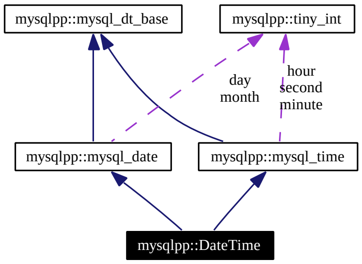
A combination of the **Date** (p. 36) and **Time** (p. 85) classes for holding MySQL DateTimes.

`#include <datetime.h>`

Inheritance diagram for mysqlpp::DateTime:



Collaboration diagram for mysqlpp::DateTime:



Public Methods

- std::ostream & **out_stream** (std::ostream &os) const

  Insert the date and time into a stream.

- cchar ∗ **convert** (cchar ∗)

  Parse a MySQL date and time string into this object.

7.16.1 Detailed Description

A combination of the **Date** (p. 36) and **Time** (p. 85) classes for holding MySQL DateTimes.

Objects of this class can be inserted into streams, and initialized from a stream.

7.16.2 Member Function Documentation

7.16.2.1 ostream & mysqlpp::DateTime::out_stream (std::ostream & *os*) const `[virtual]`

Insert the date and time into a stream.

The date and time are inserted into the stream, in that order, with a space between them.

Parameters:
  os stream to insert date and time into

Reimplemented from **mysqlpp::mysql_date** (p. 47).

The documentation for this struct was generated from the following files:

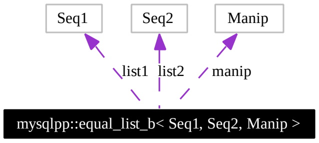
- **datetime.h**
- datetime.cpp

7.17 mysqlpp::equal_list_b< Seq1, Seq2, Manip > Struct Template Reference

Documentation needed!

`#include <vallist.h>`

Collaboration diagram for mysqlpp::equal_list_b< Seq1, Seq2, Manip >:



7.17.1 Detailed Description

template<class Seq1, class Seq2, class Manip> struct mysqlpp::equal_list_b< Seq1, Seq2, Manip >

Documentation needed!

The documentation for this struct was generated from the following file:

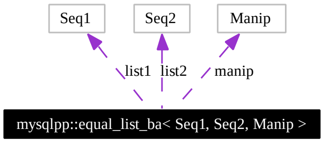
- **vallist.h**

7.18 mysqlpp::equal_list_ba< Seq1, Seq2, Manip > Struct Template Reference

Documentation needed!

`#include <vallist.h>`

Collaboration diagram for mysqlpp::equal_list_ba< Seq1, Seq2, Manip >:



7.18.1 Detailed Description

template<class Seq1, class Seq2, class Manip> struct mysqlpp::equal_list_ba< Seq1, Seq2, Manip >

Documentation needed!

The documentation for this struct was generated from the following file:

- **vallist.h**

7.19 mysqlpp::escape_type1 Struct Reference

Documentation needed!

`#include <manip.h>`

7.19.1 Detailed Description

Documentation needed!

The documentation for this struct was generated from the following file:

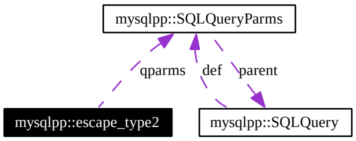
- **manip.h**

7.20 mysqlpp::escape_type2 Struct Reference

Documentation needed!

`#include <manip.h>`

Collaboration diagram for mysqlpp::escape_type2:



7.20.1 Detailed Description

Documentation needed!

The documentation for this struct was generated from the following file:

- **manip.h**

7.21 mysqlpp::FieldNames Class Reference

Holds a list of SQL field names.

`#include <field_names.h>`

Public Methods

- FieldNames & **operator=** (const **ResUse** ∗res)

  Creates a new list from the data in res.

- FieldNames & **operator=** (int i)

  Creates a new list with i field names.

- std::string & **operator**[ ] (int i)

  Get the name of a field given its index.

- uint **operator**[ ] (std::string i) const

  Get the index number of a field given its name.

7.21.1 Detailed Description

Holds a list of SQL field names.

The documentation for this class was generated from the following files:

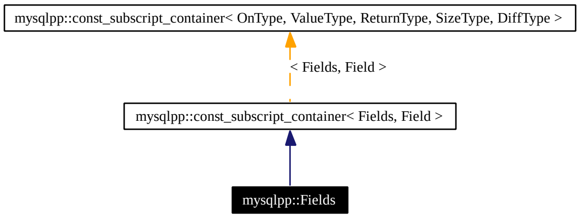
- **field_names.h**
- field_names.cpp

7.22 mysqlpp::Fields Class Reference

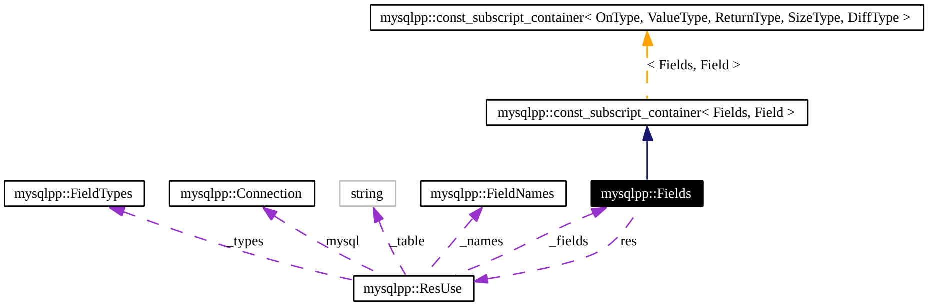
A container similar to `std::vector` for holding mysqlpp::Field records.

`#include <fields.h>`

Inheritance diagram for mysqlpp::Fields:


mysqlpp::const_subscript_container< OnType, ValueType, ReturnType, SizeType, DiffType >
< Fields, Field >
mysqlpp::const_subscript_container< Fields, Field >
mysqlpp::Fields

Collaboration diagram for mysqlpp::Fields:



Public Methods

- size_type **size** () const

  get the number of fields

- const Field & **operator**[ ] (size_type i) const

  Returns a field given its index.

7.22.1 Detailed Description

A container similar to `std::vector` for holding mysqlpp::Field records.

The documentation for this class was generated from the following files:

- **fields.h**
- fields.cpp

7.23 mysqlpp::FieldTypes Class Reference

A vector of SQL field types.

`#include <field_types.h>`

Public Methods

- FieldTypes & **operator=** (const **ResUse** ∗res)

  Creates a new list based on the info in res.

- FieldTypes & **operator=** (int i)

  Creates a new list with a given number of fields.

- **mysql_type_info** & **operator**[ ] (int i)

  Returns a field type within the list given its index.

7.23.1 Detailed Description

A vector of SQL field types.

7.23.2 Member Function Documentation

7.23.2.1 FieldTypes& mysqlpp::FieldTypes::operator= (int *i*) `[inline]`

Creates a new list with a given number of fields.

Initializes the list from the data returned by MySQL C API function `mysql_type_info()`.

Parameters:
i size of field list to create

The documentation for this class was generated from the following files:

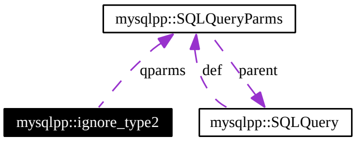
- **field_types.h**
- field_types.cpp

7.24 mysqlpp::ignore_type2 Struct Reference

Documentation needed!

`#include <manip.h>`

Collaboration diagram for mysqlpp::ignore_type2:



7.24.1 Detailed Description

Documentation needed!

The documentation for this struct was generated from the following file:

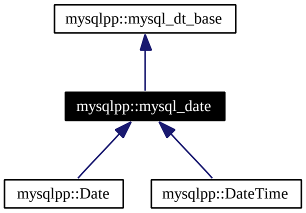
- **manip.h**

7.25 mysqlpp::mysql_date Struct Reference

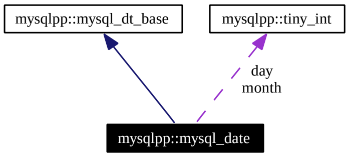
Base class of **Date** (p. 36).

`#include <datetime.h>`

Inheritance diagram for mysqlpp::mysql_date:



Collaboration diagram for mysqlpp::mysql_date:



Public Methods

- std::ostream & **out_stream** (std::ostream &os) const

  Insert the date into a stream.

- cchar ∗ **convert** (cchar ∗)

  Parse a MySQL date string into this object.

7.25.1 Detailed Description

Base class of **Date** (p. 36).

7.25.2 Member Function Documentation

7.25.2.1 ostream & mysqlpp::mysql_date::out_stream (std::ostream & *os*) const `[virtual]`

Insert the date into a stream.

The date is inserted into the stream in a format that the MySQL server can accept.

Parameters:

os stream to insert date into

Implements **mysqlpp::mysql_dt_base** (p. 49).

Reimplemented in **mysqlpp::DateTime** (p. 37).

The documentation for this struct was generated from the following files:

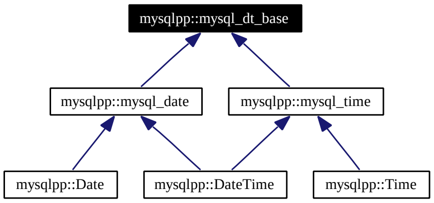
- **datetime.h**
- datetime.cpp

7.26 mysqlpp::mysql_dt_base Struct Reference

Base class for **mysql_date** (p. 47) and **mysql_time** (p. 50).

`#include <datetime.h>`

Inheritance diagram for mysqlpp::mysql_dt_base:



7.26.1 Detailed Description

Base class for **mysql_date** (p. 47) and **mysql_time** (p. 50).

The documentation for this struct was generated from the following file:

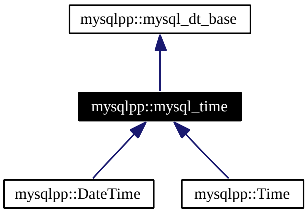
- **datetime.h**

7.27 mysqlpp::mysql_time Struct Reference

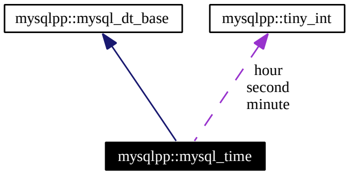
Base class of **Time** (p. 85).

`#include <datetime.h>`

Inheritance diagram for mysqlpp::mysql_time:



Collaboration diagram for mysqlpp::mysql_time:



Public Methods

- std::ostream & **out_stream** (std::ostream &os) const

  Insert the time into a stream.

- cchar * **convert** (cchar *)

  Parse a MySQL time string into this object.

7.27.1 Detailed Description

Base class of **Time** (p. 85).

7.27.2 Member Function Documentation

7.27.2.1 ostream & mysqlpp::mysql_time::out_stream (std::ostream & *os*) const `[virtual]`

Insert the time into a stream.

The time is inserted into the stream in a format that the MySQL server can accept.

Parameters:

os stream to insert time into

Implements **mysqlpp::mysql_dt_base** (p. 49).

Reimplemented in **mysqlpp::DateTime** (p. 37).

The documentation for this struct was generated from the following files:

- **datetime.h**
- datetime.cpp

7.28 mysqlpp::mysql_type_info Class Reference

Holds basic type information for ColData.

`#include <type_info.h>`

Public Methods

- const char * **name** () const

  Returns an implementation-defined name of the C++ type.

- const char * **sql_name** () const

  Returns the name of the SQL type.

- const std::type_info & **c_type** () const

  Returns the type_info for the C++ type associated with the SQL type.

- const mysql_type_info **base_type** () const

  Returns the type_info for the C++ type inside of the **mysqlpp::Null** (p. 56) *type.*

- int **id** () const

  Returns the ID of the SQL type.

- bool **quote_q** () const

  Returns true if the SQL type is of a type that needs to be quoted.

- bool **escape_q** () const

  Returns true if the SQL type is of a type that needs to be escaped.

- bool **before** (mysql_type_info &b)

  Provides a way to compare two types for sorting.

7.28.1 Detailed Description

Holds basic type information for ColData.

Class to hold basic type information for **mysqlpp::ColData** (p. 14).

7.28.2 Member Function Documentation

7.28.2.1 const mysql_type_info mysqlpp::mysql_type_info::base_type () `[inline]`

Returns the type_info for the C++ type inside of the **mysqlpp::Null** (p. 56) type.

Returns the type_info for the C++ type inside the **mysqlpp::Null** (p. 56) type. If the type is not **Null** (p. 56) then this is the same as **c_type()** (p. 53).

7.28.2.2 bool mysqlpp::mysql_type_info::before (mysql_type_info & *b*) `[inline]`

Provides a way to compare two types for sorting.

Returns true if the SQL ID of this type is lower than that of another. Used by **mysqlpp::type_info_cmp** (p. 87) when comparing types.

7.28.2.3 const std::type_info & mysqlpp::mysql_type_info::c_type () `[inline]`

Returns the type_info for the C++ type associated with the SQL type.

Returns the C++ type_info record corresponding to the SQL type.

7.28.2.4 bool mysqlpp::mysql_type_info::escape_q ()

Returns true if the SQL type is of a type that needs to be escaped.

Returns:
true if the type needs to be escaped for syntactically correct SQL.

7.28.2.5 int mysqlpp::mysql_type_info::id () const `[inline]`

Returns the ID of the SQL type.

Returns the ID number MySQL uses for this type. Note: Do not depend on the value of this ID as it may change between MySQL versions.

7.28.2.6 const char * mysqlpp::mysql_type_info::name () `[inline]`

Returns an implementation-defined name of the C++ type.

Returns the name that would be returned by typeid().**name()** (p. 53) for the C++ type associated with the SQL type.

7.28.2.7 bool mysqlpp::mysql_type_info::quote_q ()

Returns true if the SQL type is of a type that needs to be quoted.

Returns:
true if the type needs to be quoted for syntactically correct SQL.

7.28.2.8 const char * mysqlpp::mysql_type_info::sql_name () `[inline]`

Returns the name of the SQL type.

Returns the SQL name for the type.

The documentation for this class was generated from the following files:

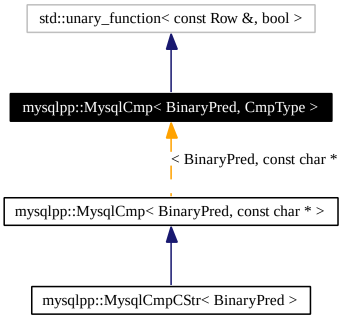
- **type_info.h**
- type_info.cpp

7.29 mysqlpp::MysqlCmp< BinaryPred, CmpType > Class Template Reference

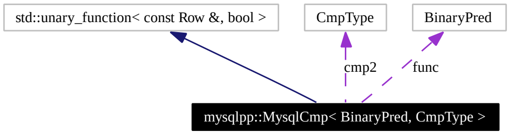
Documentation needed!

`#include <compare.h>`

Inheritance diagram for mysqlpp::MysqlCmp< BinaryPred, CmpType >:


std::unary_function< const Row &, bool >
mysqlpp::MysqlCmp< BinaryPred, CmpType >
< BinaryPred, const char * >
mysqlpp::MysqlCmp< BinaryPred, const char * >
mysqlpp::MysqlCmpCStr< BinaryPred >

Collaboration diagram for mysqlpp::MysqlCmp< BinaryPred, CmpType >:


std::unary_function< const Row &, bool >
CmpType
BinaryPred
cmp2
func
mysqlpp::MysqlCmp< BinaryPred, CmpType >

7.29.1 Detailed Description

template<class BinaryPred, class CmpType> class mysqlpp::MysqlCmp< BinaryPred, CmpType >

Documentation needed!

The documentation for this class was generated from the following file:

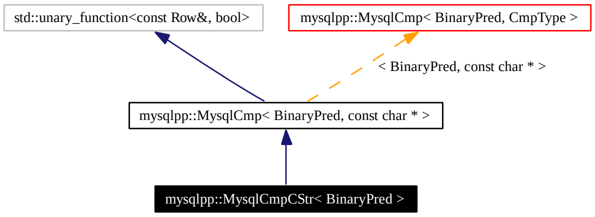
- **compare.h**

7.30 mysqlpp::MysqlCmpCStr< BinaryPred > Class Template Reference

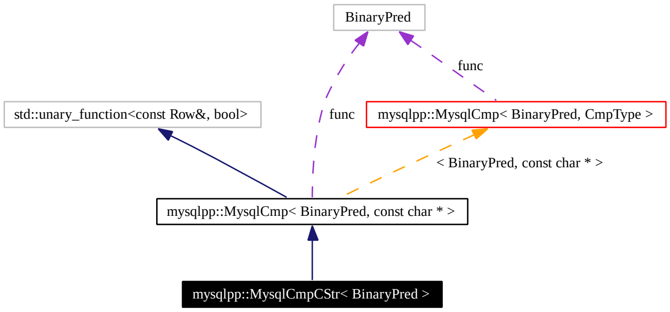
Documentation needed!

`#include <compare.h>`

Inheritance diagram for mysqlpp::MysqlCmpCStr< BinaryPred >:


std::unary_function<const Row&, bool>
mysqlpp::MysqlCmp< BinaryPred, CmpType >
< BinaryPred, const char * >
mysqlpp::MysqlCmp< BinaryPred, const char * >
mysqlpp::MysqlCmpCStr< BinaryPred >

Collaboration diagram for mysqlpp::MysqlCmpCStr< BinaryPred >:


BinaryPred
func
func
std::unary_function<const Row&, bool>
mysqlpp::MysqlCmp< BinaryPred, CmpType >
< BinaryPred, const char * >
mysqlpp::MysqlCmp< BinaryPred, const char * >
mysqlpp::MysqlCmpCStr< BinaryPred >

Public Methods

- bool **operator()** (const **Row** &cmp1) const

7.30.1 Detailed Description

template<class BinaryPred> class mysqlpp::MysqlCmpCStr< BinaryPred >

Documentation needed!

The documentation for this class was generated from the following file:

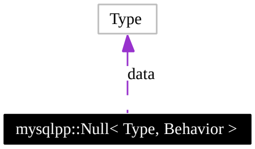
- **compare.h**

7.31 mysqlpp::Null< Type, Behavior > Class Template Reference

Container class for holding SQL nulls.

`#include <null.h>`

Collaboration diagram for mysqlpp::Null< Type, Behavior >:



Public Methods

- **Null** (const **null_type** &n)

  Gives **Null** (p. 56) *the null value.*

7.31.1 Detailed Description

template<class Type, class Behavior = NullisNull> class mysqlpp::Null< Type, Behavior >

Container class for holding SQL nulls.

This template is necessary because there is nothing in the C++ type system with the same semantics as SQL's null. (No, NULL from stddef.h is not the same!)

7.31.2 Constructor & Destructor Documentation

7.31.2.1 template<class Type, class Behavior = NullisNull> mysqlpp::Null< Type, Behavior >::Null (const null_type & *n*) `[inline]`

Gives **Null** (p. 56) the null value.

The global const `null` (not to be confused with C's NULL type) is of type **null_type** (p. 57), so you can say something like:

```
Null<Type> foo = null;
```

The documentation for this class was generated from the following file:

- **null.h**

7.32 mysqlpp::null_type Class Reference

The type of the global mysqlpp::null object.

`#include <null.h>`

7.32.1 Detailed Description

The type of the global mysqlpp::null object.

This class is for internal use only. Normal code should use **Null** (p. 56) instead.

The documentation for this class was generated from the following file:

- **null.h**

7.33 mysqlpp::NullisBlank Struct Reference

Used for the behavior parameter for template **Null** (p. 56).

`#include <null.h>`

7.33.1 Detailed Description

Used for the behavior parameter for template **Null** (p. 56).

The documentation for this struct was generated from the following file:

- **null.h**

7.34 mysqlpp::NullisNull Struct Reference

Used for the behavior parameter for template **Null** (p. 56).

`#include <null.h>`

7.34.1 Detailed Description

Used for the behavior parameter for template **Null** (p. 56).

The documentation for this struct was generated from the following file:

- **null.h**

7.35 mysqlpp::NullisZero Struct Reference

Used for the behavior parameter for template **Null** (p. 56).

`#include <null.h>`

7.35.1 Detailed Description

Used for the behavior parameter for template **Null** (p. 56).

The documentation for this struct was generated from the following file:

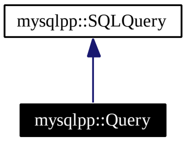
- **null.h**

7.36 mysqlpp::Query Class Reference

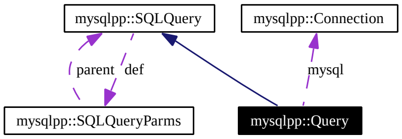
A class for building and executing SQL queries.

`#include <query.h>`

Inheritance diagram for mysqlpp::Query:



Collaboration diagram for mysqlpp::Query:



Public Methods

- **Query** (**Connection** ∗c, bool te=false)

  Create a new query object attached to a connection.

- **Query** (const Query &q)

  Create a new query object as a copy of another.

- std::string **error** ()

  Get the last error message that happened on the connection we're bound to.

- bool **success** ()

  Returns true if the query executed successfully.

- std::string **preview** ()

  Return the query string currently in the buffer.

- bool **exec** (const std::string &str)

  Execute an SQL query.

- **ResNSel execute** ()

  Execute a query and returns status information about the results of the query.

- **ResUse use** ()

  Execute a query returning data, when you wish to deal with the data one row at a time.

- **Result store** ()

  Execute a query, and store the entire result set immediately in memory.

- template<class T1> void **storein_sequence** (T1 &con, **query_reset** r=RESET_QUERY)

  Execute a query, and store the entire result set in the given STL sequence container.

- template<class T1> void **storein_set** (T1 &con, **query_reset** r=RESET_QUERY)

  Execute a query, and store the entire result set in the given associative STL container.

- template<class T1> void **storein** (T1 &con, **query_reset** r=RESET_QUERY)

  Execute a query, and store the entire result set in an STL container.

- template<class T> Query & **update** (const T &o, const T &n)

  Replace an existing row's data with new data.

- template<class T> Query & **insert** (const T &v)

  Insert a new row.

- template<class T> Query & **replace** (const T &v)

  Insert new row unless there is an existing row that matches on a unique index, in which case we replace it.

7.36.1 Detailed Description

A class for building and executing SQL queries.

This class is derived from **SQLQuery** (p. 78). It adds to that a tie between the query object and a MySQL++ **Connection** (p. 24) object, so that the query can be sent to the MySQL server we're connected to.

7.36.2 Constructor & Destructor Documentation

7.36.2.1 mysqlpp::Query::Query (Connection * *c*, bool *te* = false) `[inline]`

Create a new query object attached to a connection.

This is the constructor used by **mysqlpp::Connection::query()** (p. 27).

Parameters:

c connection the finished query should be sent out on

te if true, throw exceptions on errors

7.36.3 Member Function Documentation

7.36.3.1 bool mysqlpp::Query::exec (const std::string & *str*)

Execute an SQL query.

Use this method for queries that do not return data, and for which you only require a success or failure indication. See **execute()** (p. 63) if you need more information about the status of a query. See **store()** (p. 64), **storein()** (p. 64), and **use()** (p. 65) for alternative query execution methods.

This method is basically a thin wrapper around the MySQL C API function `mysql_query()`.

Parameters:

str the query to execute

Returns:

true if query was executed successfully

7.36.3.2 ResNSel mysqlpp::Query::execute () `[inline]`

Execute a query and returns status information about the results of the query.

Use this method for queries that do not return data as such. (For example, CREATE INDEX, or OPTIMIZE TABLE.) See **exec()** (p. 62), **store()** (p. 64), **storein()** (p. 64), and **use()** (p. 65) for alternative query execution methods.

There are a number of overloaded versions of this function. The one without parameters simply executes a query that you have built up in the object in some way. (For instance, via the **insert()** (p. 63) method, or by using the object's stream interface.) You can also pass the function an std::string containing a SQL query, a **SQLQueryParms** (p. 81) object, or as many as 12 SQLStrings. The latter two (or is it 13?) overloads are for filling out template queries.

See **exec()** (p. 62), **store()** (p. 64), **storein_sequence()** (p. 64), **storein_set()** (p. 65) and **use()** (p. 65) for alternate query execution mechanisms.

Returns:

mysqlpp::ResNSel (p. 66) object containing the state information resulting from the query

7.36.3.3 template<class T> Query& mysqlpp::Query::insert (const T & *v*) `[inline]`

Insert a new row.

This function builds an INSERT SQL query. One uses it with MySQL++'s Specialized SQL Structures mechanism.

Parameters:

v new row

See also:

replace() (p. 63), **update()** (p. 65)

Reimplemented from **mysqlpp::SQLQuery** (p. 78).

7.36.3.4 template<class T> Query& mysqlpp::Query::replace (const T & *v*) `[inline]`

Insert new row unless there is an existing row that matches on a unique index, in which case we replace it.

This function builds a REPLACE SQL query. One uses it with MySQL++'s Specialized SQL Structures mechanism.

Parameters:

v new row

See also:
insert() (p. 63), **update()** (p. 65)

Reimplemented from **mysqlpp::SQLQuery** (p. 78).

7.36.3.5 ResUse mysqlpp::Query::store () `[inline]`

Execute a query, and store the entire result set immediately in memory.

"Store" queries tell the server to send you the result set as a single block of data. (The name comes from the MySQL C API function that initiates this process, `mysql_store_result()` .) Compared to "use" queries, this mechanism keeps your code simpler, and minimizes the amount of resources that the database server has to keep tied up for you. The downside is, it can cause memory problems if the result set is sufficiently large.

See the documentation for **execute()** (p. 63) regarding the various overloads. This function has the same set of overloads.

See **exec()** (p. 62), **execute()** (p. 63), **storein()** (p. 64), and **use()** (p. 65) for alternative query execution mechanisms.

7.36.3.6 template<class T1> void mysqlpp::Query::storein (T1 & *con*, query_reset *r* = RESET_QUERY) `[inline]`

Execute a query, and store the entire result set in an STL container.

This is a set of specialized template functions that call either **storein_sequence()** (p. 64) or **storein_set()** (p. 65), depending on the type of container you pass it. It understands `std::vector`, `deque`, `list`, `slist` (a common C++ library extension), `set`, and `multiset`.

Like the functions it wraps, this is actually an overloaded set of functions. See the other functions' documentation for details.

Use this function if you think you might someday switch your program from using a set-associative container to a sequence container for storing result sets, or vice versa.

See **exec()** (p. 62), **execute()** (p. 63), **store()** (p. 64), and **use()** (p. 65) for alternative query execution mechanisms.

7.36.3.7 template<class T1> void mysqlpp::Query::storein_sequence (T1 & *con*, query_reset *r* = RESET_QUERY) `[inline]`

Execute a query, and store the entire result set in the given STL sequence container.

This is a sort of marriage between "use" queries and "store" queries. It is implemented with a "use" query, but it immediately consumes the results in a loop to populate the given sequence container. (For example, an `std::vector.`)

There are many overloads for this function, pretty much the same as for **execute()** (p. 63), except that there is a Container parameter at the front of the list. So, you can pass a container and a query string, or a container and template query parameters.

See **exec()** (p. 62), **execute()** (p. 63), **store()** (p. 64), and **use()** (p. 65) for alternative query execution mechanisms.

Parameters:
con any STL sequence container, such as `std::vector`

r whether the query automatically resets after being used

7.36.3.8 template<class T1> void mysqlpp::Query::storein_set (T1 & *con*, query_reset *r* = RESET_QUERY) [inline]

Execute a query, and store the entire result set in the given associative STL container.

The same thing as **storein_sequence()** (p. 64), except that it's used with associative STL containers, such as `std::set`.

7.36.3.9 template<class T> Query& mysqlpp::Query::update (const T & *o*, const T & *n*) [inline]

Replace an existing row's data with new data.

This function builds an UPDATE SQL query using the new row data for the SET clause, and the old row data for the WHERE clause. One uses it with MySQL++'s Specialized SQL Structures mechanism.

Parameters:

o old row

n new row

See also:

insert() (p. 63), **replace()** (p. 63)

Reimplemented from **mysqlpp::SQLQuery** (p. 78).

7.36.3.10 ResUse mysqlpp::Query::use () [inline]

Execute a query returning data, when you wish to deal with the data one row at a time.

"Use" queries tell the server to send you the results one row at a time. (The name comes from the MySQL C API function that initiates this process, `mysql_use_result()` .) This saves memory in the client program, but it potentially allows the client to tie up a connection with the database server until the result set is consumed.

See the documentation for **execute()** (p. 63) regarding the various overloads. This function has the same set of overloads.

See **exec()** (p. 62), **execute()** (p. 63), **store()** (p. 64) and **storein()** (p. 64) for alternative query execution mechanisms.

The documentation for this class was generated from the following files:

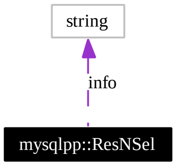
- **query.h**
- query.cpp

7.37 mysqlpp::ResNSel Class Reference

Holds the information on the success of queries that don't return any results.

`#include <result.h>`

Collaboration diagram for mysqlpp::ResNSel:



Public Methods

- **operator bool** ()

  Returns true if the query was successful.

Public Attributes

- my_ulonglong **rows**

  Number of rows affected.

- std::string **info**

  Additional info.

7.37.1 Detailed Description

Holds the information on the success of queries that don't return any results.

The documentation for this class was generated from the following files:

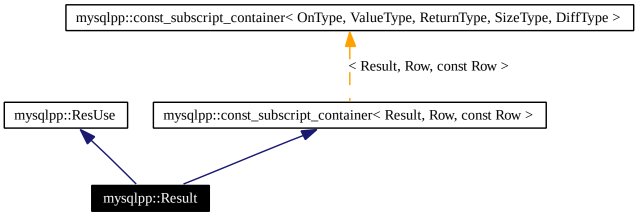
- **result.h**
- result.cpp

7.38 mysqlpp::Result Class Reference

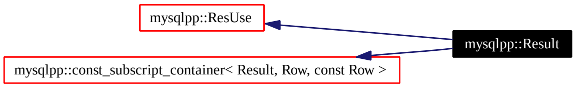
This class manages SQL result sets.

`#include <result.h>`

Inheritance diagram for mysqlpp::Result:


mysqlpp::const_subscript_container< OnType, ValueType, ReturnType, SizeType, DiffType >
< Result, Row, const Row >
mysqlpp::ResUse
mysqlpp::const_subscript_container< Result, Row, const Row >
mysqlpp::Result

Collaboration diagram for mysqlpp::Result:


mysqlpp::ResUse
mysqlpp::Result
mysqlpp::const_subscript_container< Result, Row, const Row >

Public Methods

- const **Row fetch_row** () const

  Wraps mysql_fetch_row() in MySQL C API.

- my_ulonglong **num_rows** () const

  Wraps mysql_num_rows() in MySQL C API.

- void **data_seek** (uint offset) const

  Wraps mysql_data_seek() in MySQL C API.

- size_type **size** () const

  Alias for **num_rows()** (p. 67), *only with different return type.*

- size_type **rows** () const

  Alias for **num_rows()** (p. 67), *only with different return type.*

- const **Row operator**[] (size_type i) const

  Get the row with an offset of i.

7.38.1 Detailed Description

This class manages SQL result sets.

Objects of this class are created to manage the result of "store" queries, where the result set is handed to the program as single block of row data. (The name comes from the MySQL C API function `mysql_store_result()` which creates these blocks of row data.)

This class is a random access container (in the STL sense) which is neither less-than comparable nor assignable. This container provides a reverse random-access iterator in addition to the normal forward one.

7.38.2 Member Function Documentation

7.38.2.1 const Row mysqlpp::Result::fetch_row () const `[inline]`

Wraps mysql_fetch_row() in MySQL C API.

This is simply the const version of the same function in our **parent class** (p. 69) . Why this cannot actually *be* in our parent class is beyond me.

The documentation for this class was generated from the following file:

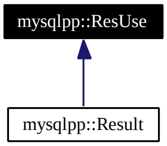
- **result.h**

7.39 mysqlpp::ResUse Class Reference

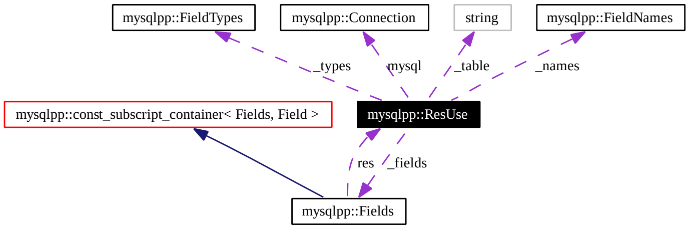
A basic result set class, for use with "use" queries.

`#include <result.h>`

Inheritance diagram for mysqlpp::ResUse:



Collaboration diagram for mysqlpp::ResUse:



Public Methods

- **Row fetch_row** ()

  Wraps mysql_fetch_row() in MySQL C API.

- bool **eof** () const

  Wraps mysql_eof() in MySQL C API.

- unsigned long ∗ **fetch_lengths** () const

  Wraps mysql_fetch_lengths() in MySQL C API.

- Field & **fetch_field** () const

  Wraps mysql_fetch_field() in MySQL C API.

- void **field_seek** (int field)

  Wraps mysql_field_seek() in MySQL C API.

- int **num_fields** () const

  Wraps mysql_num_fields() in MySQL C API.

- std::string & **table** ()

  Get the name of table that the result set comes from.

- int **field_num** (const std::string &) const

  Get the index of the named field.

- std::string & **field_name** (int)

  Get the name of the field at the given index.

- **FieldNames** & **field_names** ()

  Get the names of the fields within this result set.

- void **reset_field_names** ()

  Reset the names in the field list to their original values.

- **mysql_type_info** & **field_type** (int i)

  Get the MySQL type for a field given its index.

- **FieldTypes** & **field_types** ()

  Get a list of the types of the fields within this result set.

- void **reset_field_types** ()

  Reset the field types to their original values.

- int **names** (const std::string &s) const

  Alias for **field_num()** (p. 71).

- std::string & **names** (int i)

  Alias for **field_name()** (p. 71).

- **FieldNames** & **names** ()

  Alias for **field_names()** (p. 70).

- void **reset_names** ()

  Alias for **reset_field_names()** (p. 70).

- **mysql_type_info** & **types** (int i)

  Alias for **field_type()** (p. 70).

- **FieldTypes** & **types** ()

  Alias for **field_types()** (p. 70).

- void **reset_types** ()

  Alias for **reset_field_types()** (p. 70).

- const **Fields** & **fields** () const

  Get the underlying **Fields** (p. 44) *structure.*

- const Field & **fields** (unsigned int i) const

  Get the underlying Field structure given its index.

Protected Methods

- void **copy** (const ResUse &other)

  copy another **ResUse** (p. 69) *object's contents into this one.*

7.39.1 Detailed Description

A basic result set class, for use with "use" queries.

A "use" query is one where you make the query and then process just one row at a time in the result instead of dealing with them all as a single large chunk. (The name comes from the MySQL C API function that initiates this action, `mysql_use_result().`) By calling **fetch_row()** (p. 71) until it throws a **mysqlpp::BadQuery** (p. 22) exception (or an empty row if exceptions are disabled), you can process the result set one row at a time.

7.39.2 Member Function Documentation

7.39.2.1 void mysqlpp::ResUse::copy (const ResUse & *other*) `[protected]`

copy another **ResUse** (p. 69) object's contents into this one.

Not to be used on the self. Self-copy is not allowed.

7.39.2.2 Row mysqlpp::ResUse::fetch_row () `[inline]`

Wraps mysql_fetch_row() in MySQL C API.

This is not a thin wrapper. It does a lot of error checking before returning the **mysqlpp::Row** (p. 72) object containing the row data.

7.39.2.3 std::string & mysqlpp::ResUse::field_name (int) `[inline]`

Get the name of the field at the given index.

This is the inverse of **field_num()** (p. 71).

7.39.2.4 int mysqlpp::ResUse::field_num (const std::string &) const `[inline]`

Get the index of the named field.

This is the inverse of **field_name()** (p. 71).

The documentation for this class was generated from the following files:

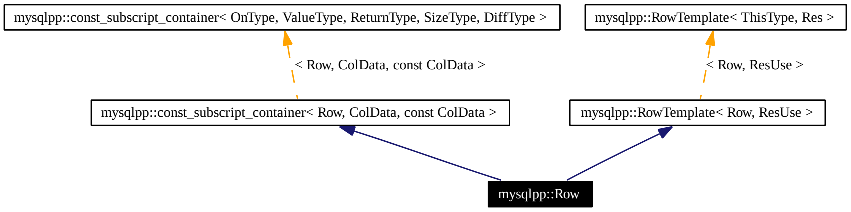
- **result.h**
- result.cpp

7.40 mysqlpp::Row Class Reference

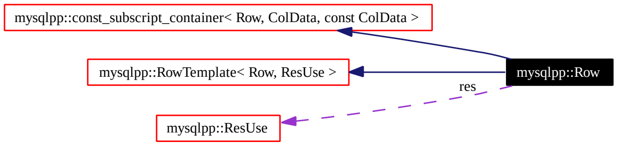
Manages rows from a result set.

`#include <row.h>`

Inheritance diagram for mysqlpp::Row:


mysqlpp::const_subscript_container< OnType, ValueType, ReturnType, SizeType, DiffType >
mysqlpp::RowTemplate< ThisType, Res >
< Row, ColData, const ColData >
< Row, ResUse >
mysqlpp::const_subscript_container< Row, ColData, const ColData >
mysqlpp::RowTemplate< Row, ResUse >
mysqlpp::Row

Collaboration diagram for mysqlpp::Row:


mysqlpp::const_subscript_container< Row, ColData, const ColData >
mysqlpp::RowTemplate< Row, ResUse >
mysqlpp::Row
res
mysqlpp::ResUse

Public Methods

- const Row & **self** () const

  Return a const reference to this object.

- Row & **self** ()

  Return a reference to this object.

- const **ResUse** & **parent** () const

  Return a reference to our parent class.

- size_type **size** () const

  Get the number of fields in the row.

- const **ColData operator**[ ] (size_type i) const

  Get the value of a field given its index.

- **operator bool** () const

  Returns true if there is data in the row.

7.40.1 Detailed Description

Manages rows from a result set.

The documentation for this class was generated from the following files:

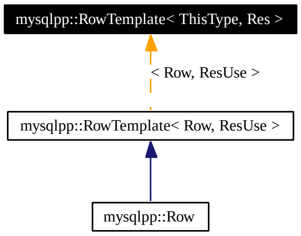
- **row.h**
- row.cpp

7.41 mysqlpp::RowTemplate< ThisType, Res > Class Template Reference

Documentation needed!

`#include <row.h>`

Inheritance diagram for mysqlpp::RowTemplate< ThisType, Res >:


mysqlpp::RowTemplate< ThisType, Res >
< Row, ResUse >
mysqlpp::RowTemplate< Row, ResUse >
mysqlpp::Row

7.41.1 Detailed Description

template<class ThisType, class Res> class mysqlpp::RowTemplate< ThisType, Res >

Documentation needed!

The documentation for this class was generated from the following file:

- **row.h**

7.42 mysqlpp::Set< Container > Class Template Reference

A special std::set derivative for holding MySQL data sets.

`#include <myset.h>`

7.42.1 Detailed Description

template<class Container = std::set<std::string>> class mysqlpp::Set< Container >

A special std::set derivative for holding MySQL data sets.

The documentation for this class was generated from the following file:

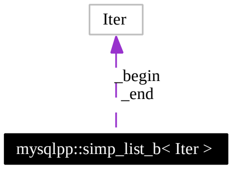
- **myset.h**

7.43 mysqlpp::simp_list_b< Iter > Class Template Reference

Documentation needed!

`#include <vallist.h>`

Collaboration diagram for mysqlpp::simp_list_b< Iter >:



7.43.1 Detailed Description

template<class Iter> class mysqlpp::simp_list_b< Iter >

Documentation needed!

The documentation for this class was generated from the following file:

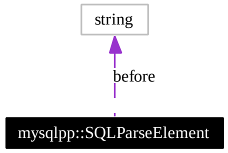
- **vallist.h**

7.44 mysqlpp::SQLParseElement Struct Reference

Documentation needed!

`#include <sql_query.h>`

Collaboration diagram for mysqlpp::SQLParseElement:



7.44.1 Detailed Description

Documentation needed!

The documentation for this struct was generated from the following file:

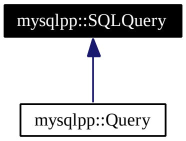
- **sql_query.h**

7.45 mysqlpp::SQLQuery Class Reference

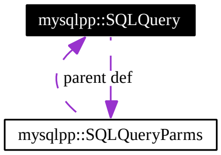
The base class for **mysqlpp::Query** (p. 61).

`#include <sql_query.h>`

Inheritance diagram for mysqlpp::SQLQuery:



Collaboration diagram for mysqlpp::SQLQuery:



Public Methods

- void **reset** ()

  Reset the query object so that it can be reused.

Public Attributes

- **SQLQueryParms def**

  The default template parameters.

Protected Methods

- void **proc** (**SQLQueryParms** &p)

  Process a parameterized query list.

7.45.1 Detailed Description

The base class for **mysqlpp::Query** (p. 61).

One uses an object of this class to form queries that can be sent to the database server via the **mysqlpp::Connection** (p. 24) object.

This class is subclassed from `std::stringstream`. This means that you can form a SQL query using C++ stream idioms without having to create your own `stringstream` object and then dump that into the query object. And of course, it gets you all the benefits of C++ streams, such as type safety, which `sprintf()` and such do not offer. Although you can read from this object as you would any other stream, this is *not* recommended. It may fail in strange ways, and there is no support offered if you break it by doing so.

If you seek within the stream in any way, be sure to reset the stream pointer to the end before calling any of the **SQLQuery** (p. 78)-specific methods except for error() and success().

7.45.2 Member Function Documentation

7.45.2.1 void mysqlpp::SQLQuery::reset ()

Reset the query object so that it can be reused.

This erases the query string and the contents of the parameterized query element list.

7.45.3 Member Data Documentation

7.45.3.1 SQLQueryParms mysqlpp::SQLQuery::def

The default template parameters.

Used for filling in parameterized queries.

The documentation for this class was generated from the following files:

- **sql_query.h**
- sql_query.cpp

7.46 mysqlpp::SQLQueryNEParms Class Reference

Exception thrown when not enough parameters are provided.

`#include <exceptions.h>`

7.46.1 Detailed Description

Exception thrown when not enough parameters are provided.

Thrown when not enough parameters are provided for a template query.

The documentation for this class was generated from the following file:

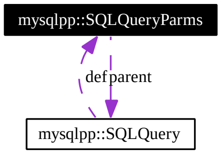
- **exceptions.h**

7.47 mysqlpp::SQLQueryParms Class Reference

This class holds the parameter values for filling template queries.

`#include <sql_query.h>`

Collaboration diagram for mysqlpp::SQLQueryParms:



Public Methods

- void **clear** ()

  Clears the list.

- **SQLString** & **operator**[ ] (size_type n)

  Access element number n.

- const **SQLString** & **operator**[ ] (size_type n) const

  Access element number n.

- **SQLString** & **operator**[ ] (const char ∗str)

  Access the value of the element with a key of str.

- const **SQLString** & **operator**[ ] (const char ∗str) const

  Access the value of the element with a key of str.

- SQLQueryParms & **operator**<< (const **SQLString** &str)

  Adds an element to the list.

- SQLQueryParms & **operator**+= (const **SQLString** &str)

  Adds an element to the list.

- void **set** (ss a)

  Set (p. 75) *the template query parameters.*

7.47.1 Detailed Description

This class holds the parameter values for filling template queries.

7.47.2 Member Function Documentation

7.47.2.1 void mysqlpp::SQLQueryParms::set (ss *a*) `[inline]`

Set (p. 75) the template query parameters.

Sets element 0 to a, element 1 to b, etc. May specify up to a dozen parameters.

The documentation for this class was generated from the following files:

- **sql_query.h**
- sql_query.cpp

7.48 mysqlpp::SQLString Class Reference

A specialized `std::string` that will convert from any valid MySQL type.

`#include <sql_string.h>`

7.48.1 Detailed Description

A specialized `std::string` that will convert from any valid MySQL type.

The documentation for this class was generated from the following files:

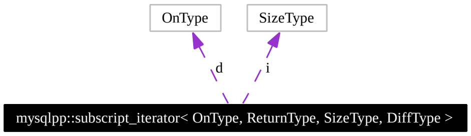
- **sql_string.h**
- sql_string.cpp

7.49 mysqlpp::subscript_iterator< OnType, ReturnType, SizeType, DiffType > Class Template Reference

Iterator that can be subscripted.

`#include <resiter.h>`

Collaboration diagram for mysqlpp::subscript_iterator< OnType, ReturnType, SizeType, DiffType >:


OnType
SizeType
d
i
mysqlpp::subscript_iterator< OnType, ReturnType, SizeType, DiffType >

7.49.1 Detailed Description

template<class OnType, class ReturnType, class SizeType, class DiffType> class mysqlpp::subscript_iterator< OnType, ReturnType, SizeType, DiffType >

Iterator that can be subscripted.

This is the type of iterator used by the **const_subscript_container** (p. 29) template.

The documentation for this class was generated from the following file:

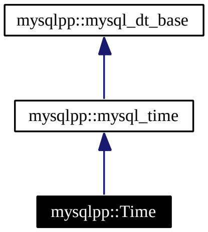
- **resiter.h**

7.50 mysqlpp::Time Struct Reference

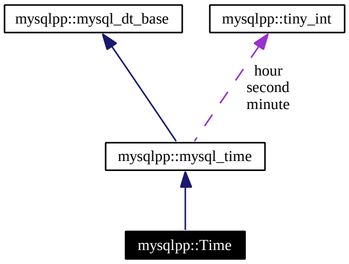
Holds MySQL times.

`#include <datetime.h>`

Inheritance diagram for mysqlpp::Time:



Collaboration diagram for mysqlpp::Time:



7.50.1 Detailed Description

Holds MySQL times.

Objects of this class can be inserted into streams, and initialized from a stream.

The documentation for this struct was generated from the following file:

- **datetime.h**

7.51 mysqlpp::tiny_int Class Reference

Class for holding an SQL `tiny_int` object.

`#include <tiny_int.h>`

7.51.1 Detailed Description

Class for holding an SQL `tiny_int` object.

This is required because the closest C++ type, `char`, doesn't have all the right semantics. For one, inserting a `char` into a stream won't give you a number.

The documentation for this class was generated from the following file:

- **tiny_int.h**

7.52 mysqlpp::type_info_cmp Struct Reference

Documentation needed!

`#include <type_info.h>`

7.52.1 Detailed Description

Documentation needed!

The documentation for this struct was generated from the following file:

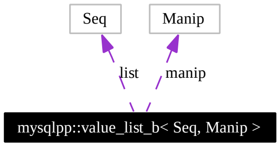
- **type_info.h**

7.53 mysqlpp::value_list_b< Seq, Manip > Struct Template Reference

Documentation needed!

`#include <vallist.h>`

Collaboration diagram for mysqlpp::value_list_b< Seq, Manip >:



7.53.1 Detailed Description

template<class Seq, class Manip> struct mysqlpp::value_list_b< Seq, Manip >

Documentation needed!

The documentation for this struct was generated from the following file:

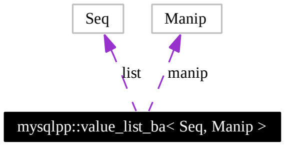
- **vallist.h**

7.54 mysqlpp::value_list_ba< Seq, Manip > Struct Template Reference

Documentation needed!

`#include <vallist.h>`

Collaboration diagram for mysqlpp::value_list_ba< Seq, Manip >:



7.54.1 Detailed Description

template<class Seq, class Manip> struct mysqlpp::value_list_ba< Seq, Manip >

Documentation needed!

The documentation for this struct was generated from the following file:

- **vallist.h**

Chapter 8

MySQL++ File Documentation

8.1 coldata.h File Reference

Declares classes for converting string data to any of the basic C types.

`#include "platform.h"`

`#include "const_string.h"`

`#include "convert.h"`

`#include "defs.h"`

`#include "exceptions.h"`

`#include "null.h"`

`#include "string_util.h"`

`#include "type_info.h"`

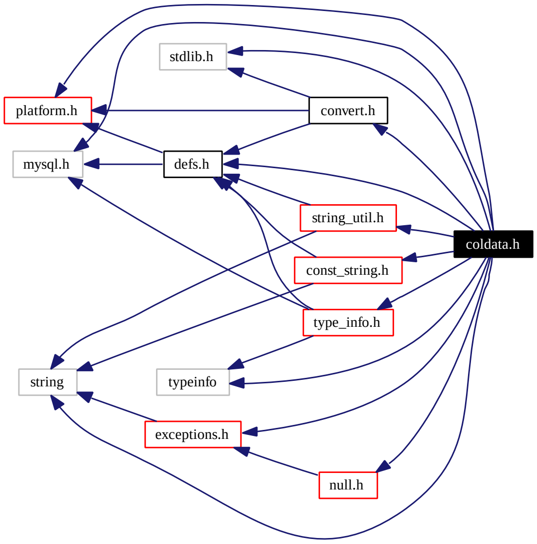
`#include <mysql.h>`

`#include <typeinfo>`

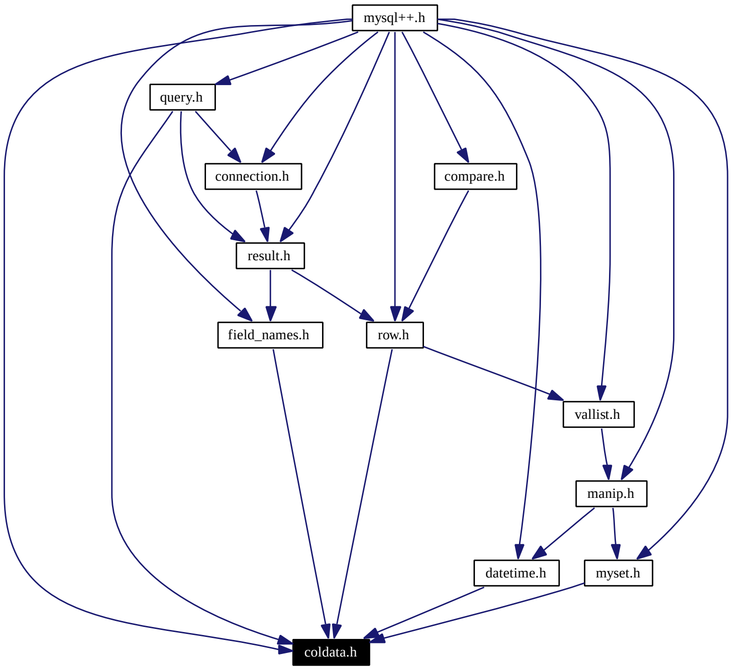
`#include <string>`

`#include <stdlib.h>`

Include dependency graph for coldata.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.1.1 Detailed Description

Declares classes for converting string data to any of the basic C types.

Roughly speaking, this defines classes that are the inverse of **mysqlpp::SQLString** (p. 83).

8.1.2 Define Documentation

8.1.2.1 #define operator_binary(other, conv)

Value:

```
oprsw(+, other, conv) \
  oprsw(-, other, conv) \
  oprsw(*, other, conv) \
  oprsw(/, other, conv)
```

8.1.2.2 #define operator_binary_int(other, conv)

Value:

```
operator_binary(other, conv) \
  oprsw(%, other, conv) \
  oprsw(&, other, conv) \
  oprsw(^, other, conv) \
  oprsw(|, other, conv) \
  oprsw(<<, other, conv) \
  oprsw(>>, other, conv)
```

8.1.2.3 #define oprsw(opr, other, conv)

Value:

```
template<class Str> \
  inline other operator opr (ColData_Tmpl<Str> x, other y) \
    {return static_cast<conv>(x) opr y;} \
  template<class Str> \
  inline other operator opr (other x, ColData_Tmpl<Str> y) \
    {return x opr static_cast<conv>(y);}
```

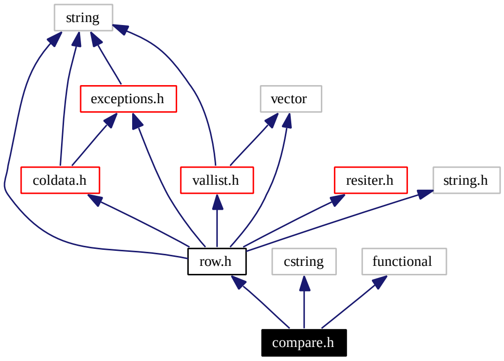
8.2 compare.h File Reference

`#include "row.h"`

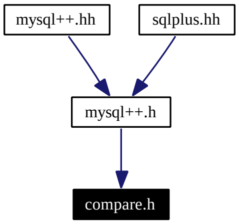
`#include <cstring>`

`#include <functional>`

Include dependency graph for compare.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.2.1 Detailed Description

Documentation needed!

8.3 connection.h File Reference

Declares the Connection class.

`#include "platform.h"`

`#include "exceptions.h"`

`#include "result.h"`

`#include <mysql.h>`

`#include <vector>`

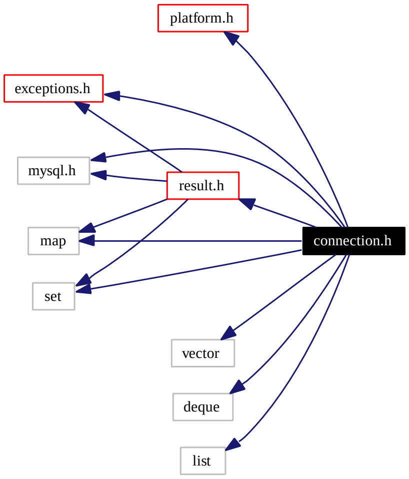
`#include <deque>`

`#include <list>`

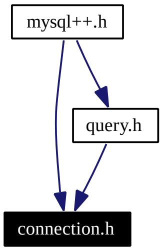
`#include <set>`

`#include <map>`

Include dependency graph for connection.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.3.1 Detailed Description

Declares the Connection class.

Every program using MySQL++ must create a Connection object, which manages information about the connection to the MySQL database. In addition, this class controls things like whether exceptions are thrown when errors are encountered.

8.4 const_string.h File Reference

Declares a wrapper for `const char*` which behaves in a way more useful to MySQL++.

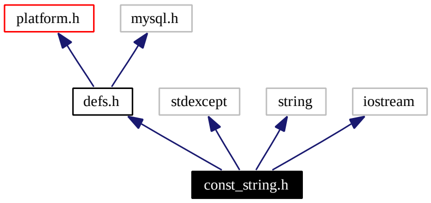
`#include "defs.h"`

`#include <stdexcept>`

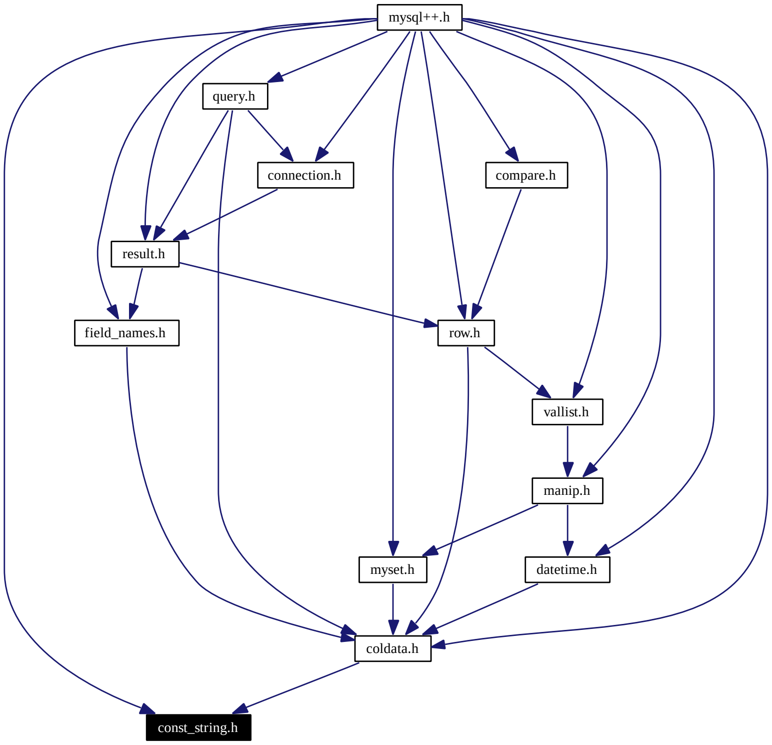
`#include <string>`

`#include <iostream>`

Include dependency graph for const_string.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.4.1 Detailed Description

Declares a wrapper for `const char*` which behaves in a way more useful to MySQL++.

8.5 convert.h File Reference

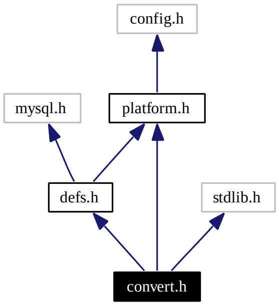
Declares various string-to-integer type conversion templates.

`#include "platform.h"`

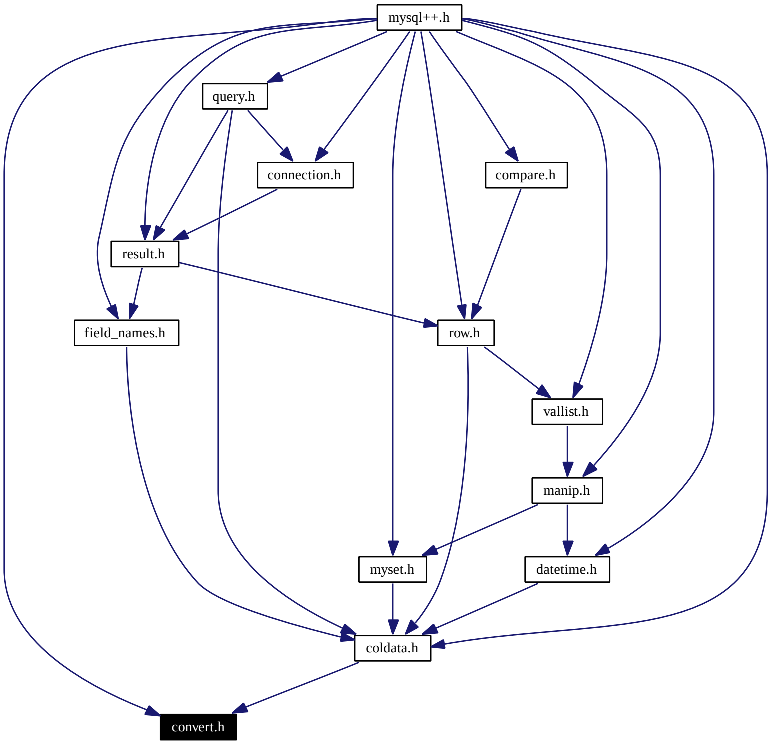
`#include "defs.h"`

`#include <stdlib.h>`

Include dependency graph for convert.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.5.1 Detailed Description

Declares various string-to-integer type conversion templates.

These templates are the mechanism used within **mysqlpp::ColData_Tmpl** (p. 23) for its string-to-*something* conversions.

8.5.2 Define Documentation

8.5.2.1 #define mysql_convert(TYPE, FUNC)

Value:

```
template <> \
  class mysql_convert<TYPE> {\
  private:\
    TYPE num;\
  public:\
    mysql_convert(const char* str, const char *& end) { \
      num = FUNC(str, const_cast<char **>(&end),10);}\
    operator TYPE () {return num;}\
  };\
```

8.5.2.2 #define mysql_convert(TYPE, FUNC)

Value:

```
template <> \
  class mysql_convert<TYPE> {\
  private:\
    TYPE num;\
  public:\
    mysql_convert(const char* str, const char *& end) { \
      num = FUNC(str, const_cast<char **>(&end));}\
    operator TYPE () {return num;}\
  };\
```

8.6 datetime.h File Reference

Declares classes to add MySQL-compatible date and time types to C++'s type system.

`#include "defs.h"`

`#include "coldata.h"`

`#include "stream2string.h"`

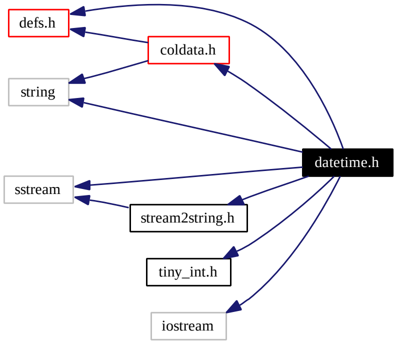
`#include "tiny_int.h"`

`#include <string>`

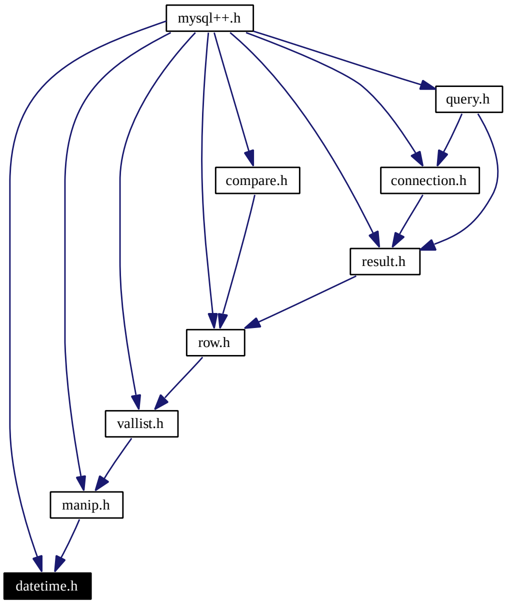
`#include <sstream>`

`#include <iostream>`

Include dependency graph for datetime.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.6.1 Detailed Description

Declares classes to add MySQL-compatible date and time types to C++'s type system.

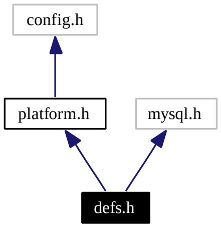
8.7 defs.h File Reference

Standard definitions used all across the library, particularly things that don't fit well anywhere else.

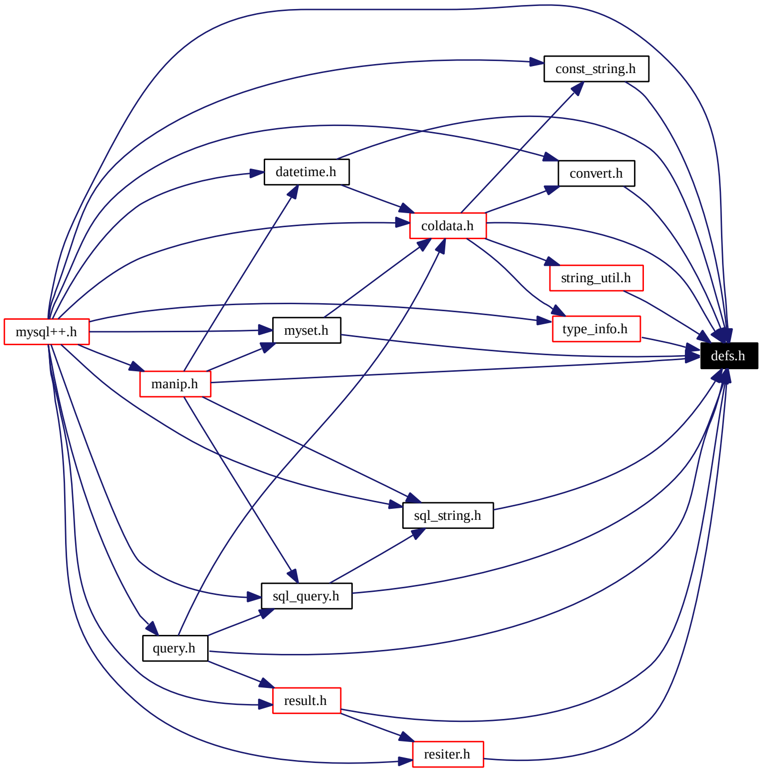
`#include "platform.h"`

`#include <mysql.h>`

Include dependency graph for defs.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.7.1 Detailed Description

Standard definitions used all across the library, particularly things that don't fit well anywhere else.

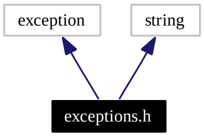
8.8 exceptions.h File Reference

Declares the MySQL++-specific exception classes.

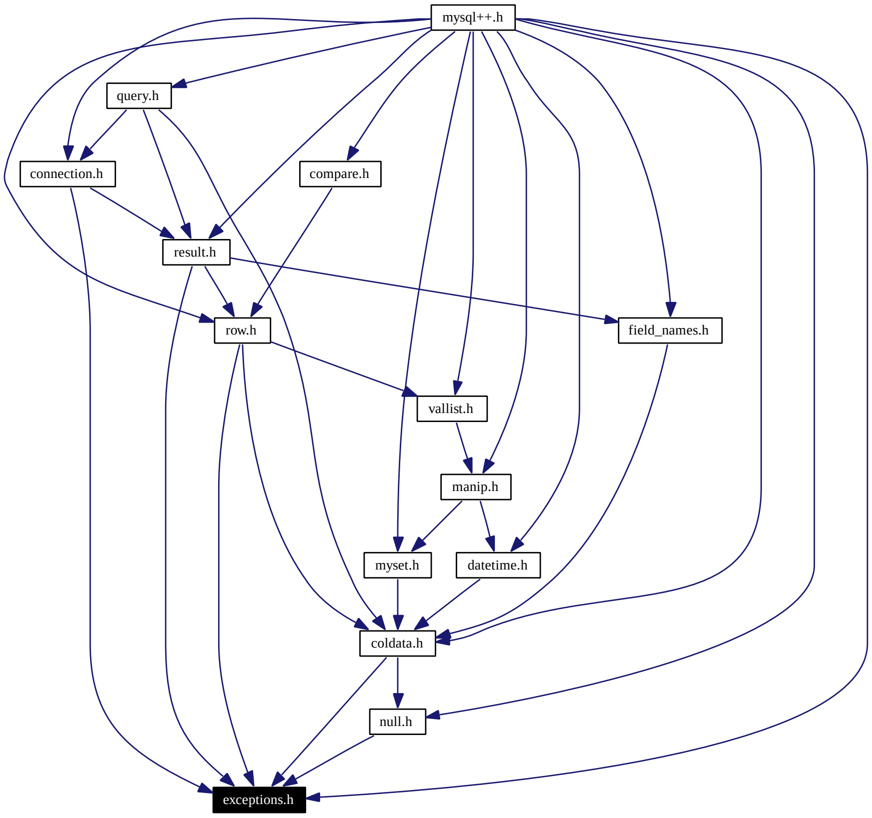
`#include <exception>`

`#include <string>`

Include dependency graph for exceptions.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.8.1 Detailed Description

Declares the MySQL++-specific exception classes.

When exceptions are enabled for a given **Connection** (p. 24) object, any of these exceptions can be thrown as a result of operations done through that connection.

8.9 field_names.h File Reference

Declares a class to hold a list of field names.

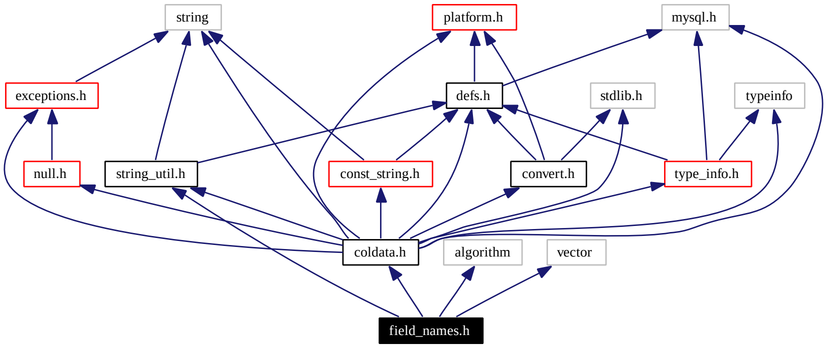
`#include "coldata.h"`

`#include "string_util.h"`

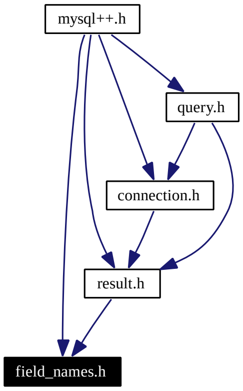
`#include <algorithm>`

`#include <vector>`

Include dependency graph for field_names.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.9.1 Detailed Description

Declares a class to hold a list of field names.

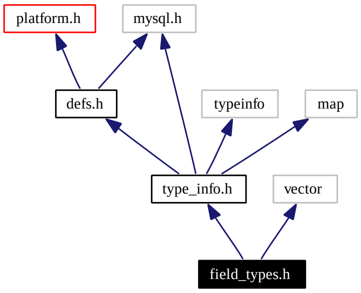
8.10 field_types.h File Reference

Declares a class to hold a list of SQL field type info.

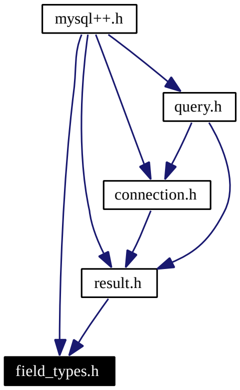
`#include "type_info.h"`

`#include <vector>`

Include dependency graph for field_types.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.10.1 Detailed Description

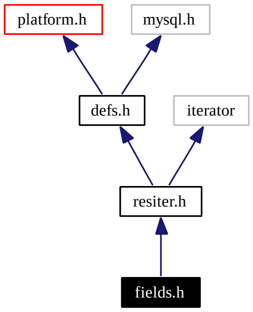
Declares a class to hold a list of SQL field type info.

8.11 fields.h File Reference

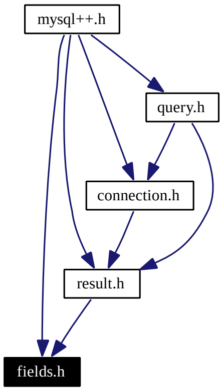
Declares a class for holding information about a set of fields.

`#include "resiter.h"`

Include dependency graph for fields.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.11.1 Detailed Description

Declares a class for holding information about a set of fields.

8.12 manip.h File Reference

Declares `std::ostream` manipulators useful with SQL syntax.

`#include "defs.h"`

`#include "datetime.h"`

`#include "myset.h"`

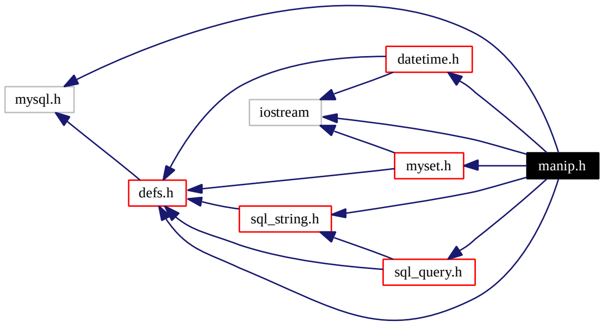
`#include "sql_string.h"`

`#include "sql_query.h"`

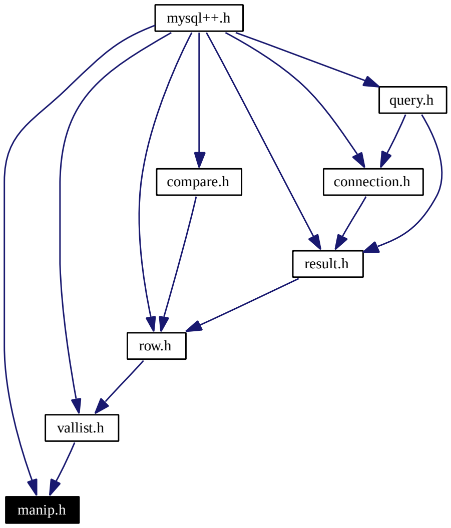
`#include <mysql.h>`

`#include <iostream>`

Include dependency graph for manip.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.12.1 Detailed Description

Declares `std::ostream` manipulators useful with SQL syntax.

These manipulators let you automatically quote elements or escape characters that are special in SQL when inserting them into an `std::ostream`. Since **mysqlpp::SQLQuery** (p. 78) is an ostream, these manipulators make it easier to build syntactically-correct SQL queries.

This file also includes `operator<<` definitions for ColData_Tmpl, one of the MySQL++ string-like classes. When inserting such items into a stream, they are automatically quoted and escaped as necessary unless the global variable dont_quote_auto is set to true. These operators are smart enough to turn this behavior off when the stream is `cout` or `cerr`, however, since quoting and escaping are surely not required in that instance.

8.13 myset.h File Reference

Declares templates for generating custom containers used elsewhere in the library.

`#include "defs.h"`

`#include "coldata.h"`

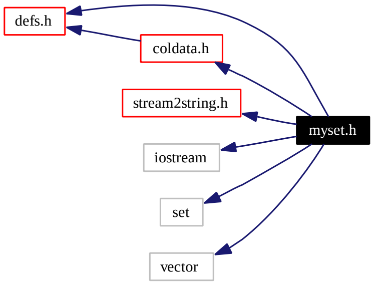
`#include "stream2string.h"`

`#include <iostream>`

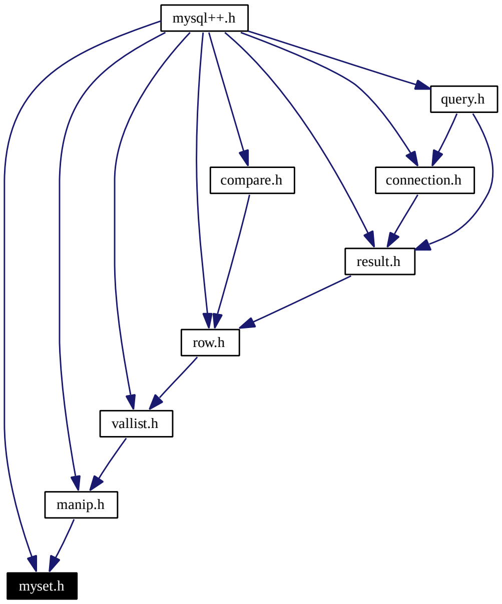
`#include <set>`

`#include <vector>`

Include dependency graph for myset.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.13.1 Detailed Description

Declares templates for generating custom containers used elsewhere in the library.

8.14 mysql++.h File Reference

The main MySQL++ header file. It simply includes all of the other header files (except for custom.h) in the proper order.

`#include "platform.h"`

`#include "defs.h"`

`#include "coldata.h"`

`#include "compare.h"`

`#include "connection.h"`

`#include "const_string.h"`

`#include "convert.h"`

`#include "datetime.h"`

`#include "exceptions.h"`

`#include "field_names.h"`

`#include "field_types.h"`

`#include "fields.h"`

`#include "manip.h"`

`#include "myset.h"`

`#include "null.h"`

`#include "query.h"`

`#include "resiter.h"`

`#include "result.h"`

`#include "row.h"`

`#include "sql_query.h"`

`#include "sql_string.h"`

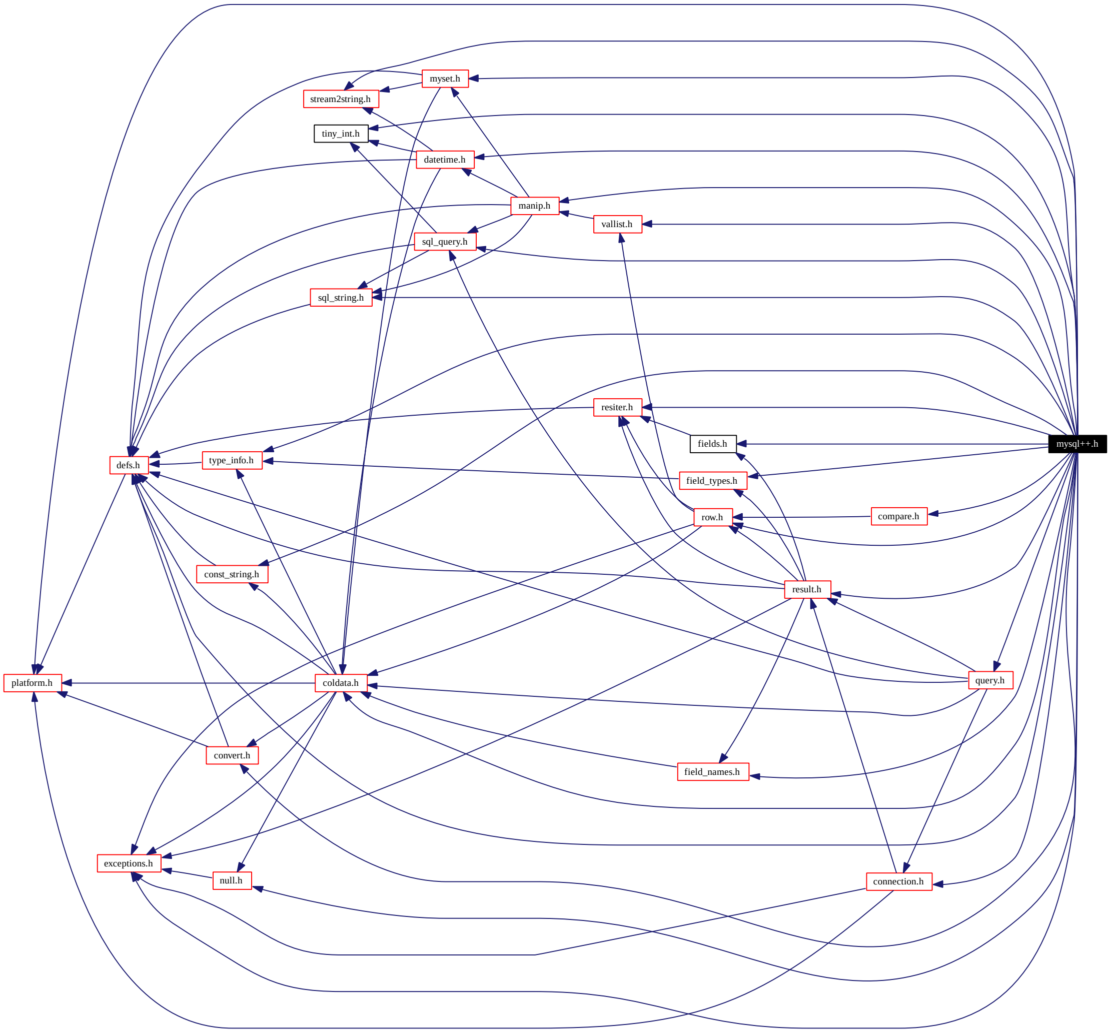
`#include "stream2string.h"`

`#include "tiny_int.h"`

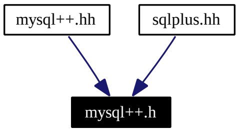
`#include "type_info.h"`

`#include "vallist.h"`

Include dependency graph for mysql++.h:



This graph shows which files directly or indirectly include this file:



8.14.1 Detailed Description

The main MySQL++ header file. It simply includes all of the other header files (except for custom.h) in the proper order.

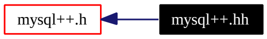
Most programs will have no reason to include any of the other MySQL++ headers directly, except for custom.h.

8.15 mysql++.hh File Reference

Deprecated backwards-compatibility header. Use **mysql++.h** in new code instead.

`#include "mysql++.h"`

Include dependency graph for mysql++.hh:



8.15.1 Detailed Description

Deprecated backwards-compatibility header. Use **mysql++.h** in new code instead.

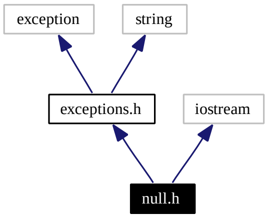
8.16 null.h File Reference

Declares classes that implement SQL "null" semantics within C++'s type system.

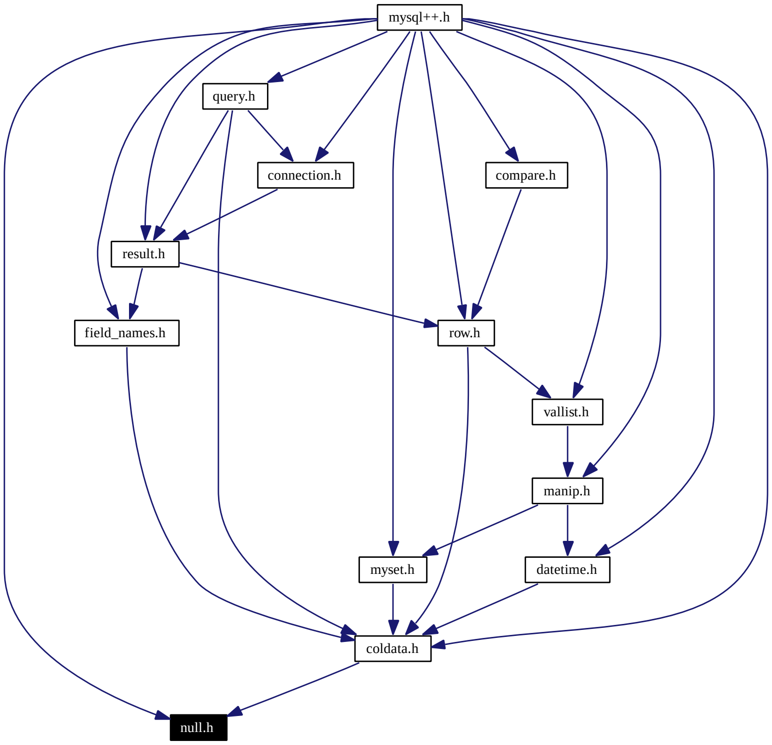
`#include "exceptions.h"`

`#include <iostream>`

Include dependency graph for null.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.16.1 Detailed Description

Declares classes that implement SQL "null" semantics within C++'s type system.

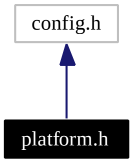
This is required because C++'s own NULL type is not semantically the same as SQL nulls.

8.17 platform.h File Reference

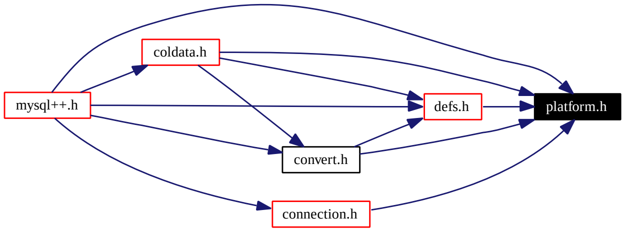
This file includes things that help the rest of MySQL++.

`#include "config.h"`

Include dependency graph for platform.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define **MYSQLPP_DLLEXPORT**

8.17.1 Detailed Description

This file includes things that help the rest of MySQL++.

8.17.2 Define Documentation

8.17.2.1 #define MYSQLPP_DLLEXPORT

This file includes all of the platform-specific definitions that allow the rest of the code to be aware only of platform features, rather than aware of specific platforms. On Unixy systems, it includes the autoconf-generated header config.h, and on all other platforms it includes the tests for platform features directly.

8.18 query.h File Reference

Defines the user-facing Query class, which is used to build up SQL queries, and execute them.

`#include "defs.h"`

`#include "coldata.h"`

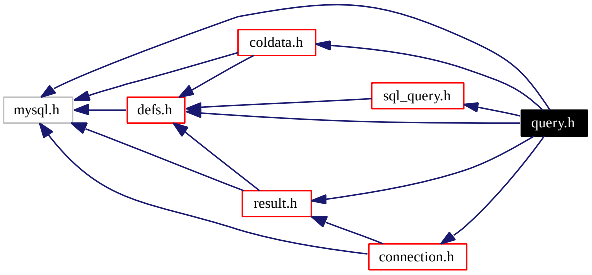
`#include "connection.h"`

`#include "result.h"`

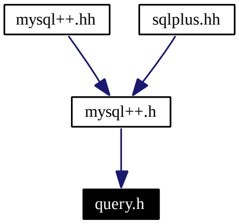
`#include "sql_query.h"`

`#include <mysql.h>`

Include dependency graph for query.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.18.1 Detailed Description

Defines the user-facing Query class, which is used to build up SQL queries, and execute them.

This class is used in one of two main fashions.

First, it has several member functions that can build specific query types. For instance, **insert()** (p. 63) allows you to build an INSERT statement for a Specialized SQL Structure.

Second, one of its base classes is std::stringstream, so you may build an SQL query in the same way as you'd build up any other string with C++ streams.

One does not generally create Query objects directly. Instead, call **mysqlpp::Connection::query()** (p. 27) to get one tied to that connection.

8.18.2 Define Documentation

8.18.2.1 #define mysql_query_define1(RETURN, FUNC)

Value:

```
RETURN FUNC (const char* str); \
  RETURN FUNC (parms &p);\
  mysql_query_define0(RETURN,FUNC) \
```

8.18.2.2 #define mysql_query_define2(FUNC)

Value:

```
template <class T1> void FUNC (T1 &con, const char* str); \
  template <class T1> void FUNC (T1 &con, parms &p, query_reset r = RESET_QUERY);\
  template <class T1> void FUNC (T1 &con, ss a)\
        {FUNC (con, parms() << a);}\
  template <class T1> void FUNC (T1 &con, ss a, ss b)\
        {FUNC (con, parms() << a << b);}\
  template <class T1> void FUNC (T1 &con, ss a, ss b, ss c)\
        {FUNC (con, parms() << a << b << c);}\
  template <class T1> void FUNC (T1 &con, ss a, ss b, ss c, ss d)\
        {FUNC (con, parms() << a << b << c << d);}\
  template <class T1> void FUNC (T1 &con, ss a, ss b, ss c, ss d, ss e)\
        {FUNC (con, parms() << a << b << c << d << e);} \
  template <class T1> void FUNC (T1 &con, ss a, ss b, ss c, ss d, ss e, ss f)\
        {FUNC (con, parms() << a << b << c << d << e << f);}\
  template <class T1> void FUNC (T1 &con,ss a,ss b,ss c,ss d,ss e,ss f,ss g)\
        {FUNC (con, parms() << a << b << c << d << e << f << g);}\
  template <class T1> void FUNC (T1 &con,ss a,ss b,ss c,ss d,ss e,ss f,ss g,ss h)\
        {FUNC (con, parms() << a << b << c << d << e << f << g << h);}\
  template <class T1> void FUNC (T1 &con, ss a, ss b, ss c, ss d, ss e, ss f, ss g, ss h, ss i)\
        {FUNC (con, parms() << a << b << c << d << e << f << g << h << i);}\
  template <class T1> void FUNC (T1 &con, ss a,ss b,ss c,ss d,ss e,ss f,ss g,ss h,ss i,ss j)\
        {FUNC (con, parms() <<a <<b <<c <<d <<e <<f <<g <<h <<i <<j);}\
  template <class T1> void FUNC (T1 &con, ss a,ss b,ss c,ss d,ss e,ss f,ss g,ss h,ss i,ss j,ss k)\
        {FUNC (con, parms() <<a <<b <<c <<d <<e <<f <<g <<h <<i <<j <<k);}\
  template <class T1> void FUNC (T1 &con, ss a,ss b,ss c,ss d,ss e,ss f,ss g,ss h,ss i,ss j,ss k,\
                           ss l)\
        {FUNC (con, parms() <<a <<b <<c <<d <<e <<f <<g <<h <<i <<j <<k <<l);}\
```

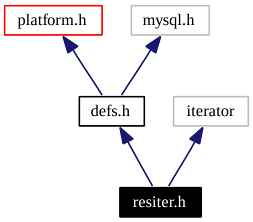
8.19 resiter.h File Reference

Declares templates for adapting existing classes to be iteratable random-access containers.

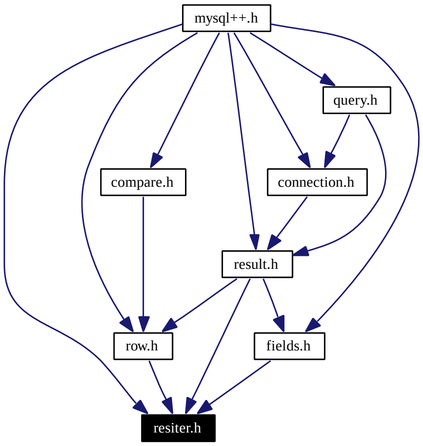
`#include "defs.h"`

`#include <iterator>`

Include dependency graph for resiter.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.19.1 Detailed Description

Declares templates for adapting existing classes to be iteratable random-access containers.

The file name seems to tie it to the **mysqlpp::Result** (p. 67) class, which is so adapted, but these templates are also used to adapt the **mysqlpp::Fields** (p. 44) and **mysqlpp::Row** (p. 72) classes.

8.20 result.h File Reference

Declares classes for holding SQL query result sets.

`#include "defs.h"`

`#include "exceptions.h"`

`#include "fields.h"`

`#include "field_names.h"`

`#include "field_types.h"`

`#include "resiter.h"`

`#include "row.h"`

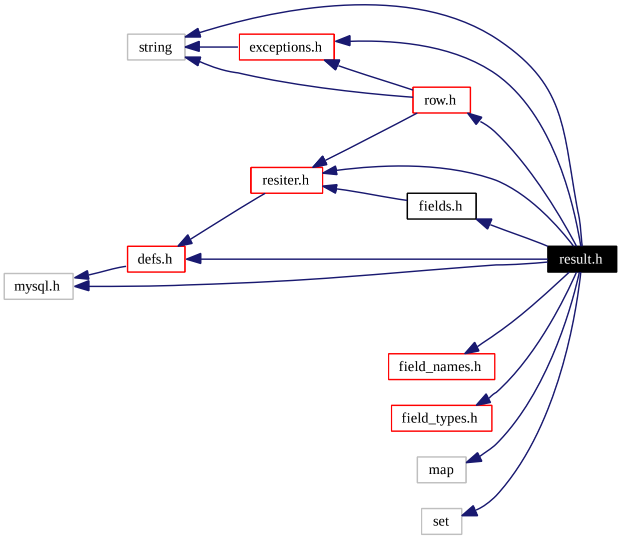
`#include <mysql.h>`

`#include <map>`

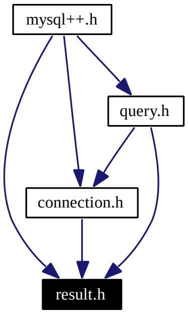
`#include <set>`

`#include <string>`

Include dependency graph for result.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.20.1 Detailed Description

Declares classes for holding SQL query result sets.

8.21 row.h File Reference

Declares the classes for holding row data from a result set.

`#include "coldata.h"`

`#include "exceptions.h"`

`#include "resiter.h"`

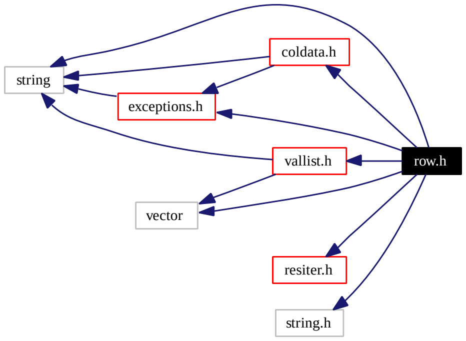
`#include "vallist.h"`

`#include <vector>`

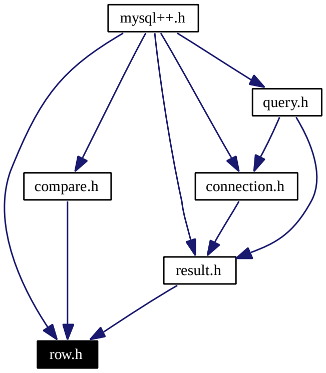
`#include <string>`

`#include <string.h>`

Include dependency graph for row.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.21.1 Detailed Description

Declares the classes for holding row data from a result set.

8.22 sql_query.h File Reference

Declares the base class for **mysqlpp::Query** (p. 61), plus some utility classes to be used with it.

`#include "defs.h"`

`#include "sql_string.h"`

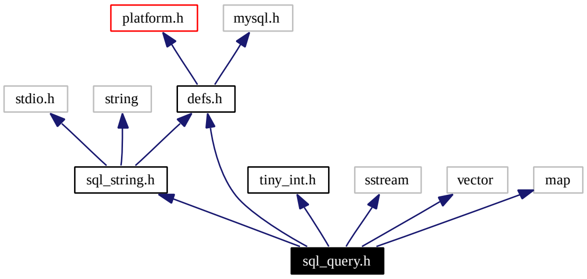
`#include "tiny_int.h"`

`#include <sstream>`

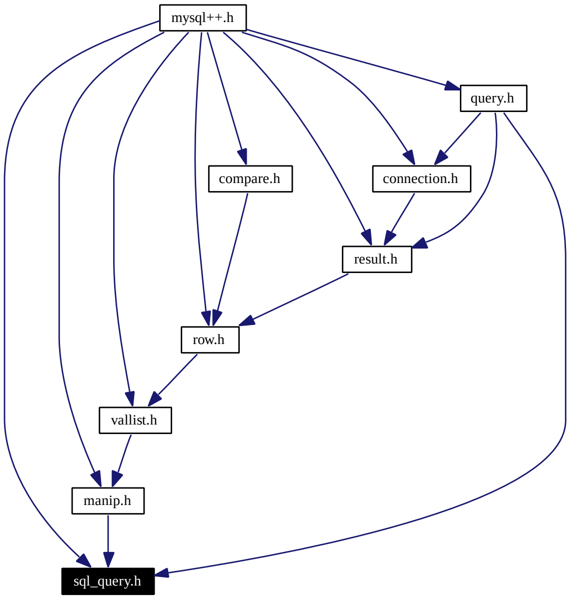
`#include <vector>`

`#include <map>`

Include dependency graph for sql_query.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.22.1 Detailed Description

Declares the base class for **mysqlpp::Query** (p. 61), plus some utility classes to be used with it.

Class SQLQuery contains a large part of the functionality of class Query, which is the only thing that derives from this class. It is separate for historical reasons only: early on, there was a dream (and some effort) to make MySQL++ database-independent. Once maintainership shifted to MySQL AB employees in 1999, though, that dream died.

The current maintainers have no wish to try and revive that dream, so at some point this class's contents will be folded into the Query class. This will probably happen in the next major release, when major ABI breakage is acceptable.

8.22.2 Define Documentation

8.22.2.1 #define mysql_query_define0(RETURN, FUNC)

Value:

```
RETURN FUNC (ss a)\
    {return FUNC (parms() << a);}\
  RETURN FUNC (ss a, ss b)\
    {return FUNC (parms() << a << b);}\
  RETURN FUNC (ss a, ss b, ss c)\
    {return FUNC (parms() << a << b << c);}\
  RETURN FUNC (ss a, ss b, ss c, ss d)\
    {return FUNC (parms() << a << b << c << d);}\
  RETURN FUNC (ss a, ss b, ss c, ss d, ss e)\
    {return FUNC (parms() << a << b << c << d << e);} \
  RETURN FUNC (ss a, ss b, ss c, ss d, ss e, ss f)\
    {return FUNC (parms() << a << b << c << d << e << f);}\
  RETURN FUNC (ss a, ss b, ss c, ss d, ss e, ss f, ss g)\
    {return FUNC (parms() << a << b << c << d << e << f << g);}\
  RETURN FUNC (ss a, ss b, ss c, ss d, ss e, ss f, ss g, ss h)\
    {return FUNC (parms() << a << b << c << d << e << f << g << h);}\
  RETURN FUNC (ss a, ss b, ss c, ss d, ss e, ss f, ss g, ss h, ss i)\
    {return FUNC (parms() << a << b << c << d << e << f << g << h << i);}\
  RETURN FUNC (ss a,ss b,ss c,ss d,ss e,ss f,ss g,ss h,ss i,ss j)\
    {return FUNC (parms() <<a <<b <<c <<d <<e <<f <<g <<h <<i <<j);}\
  RETURN FUNC (ss a,ss b,ss c,ss d,ss e,ss f,ss g,ss h,ss i,ss j,ss k)\
    {return FUNC (parms() <<a <<b <<c <<d <<e <<f <<g <<h <<i <<j <<k);}\
  RETURN FUNC (ss a,ss b,ss c,ss d,ss e,ss f,ss g,ss h,ss i,ss j,ss k,\
                       ss l)\
    {return FUNC (parms() <<a <<b <<c <<d <<e <<f <<g <<h <<i <<j <<k <<l);}\
```

8.23 sql_string.h File Reference

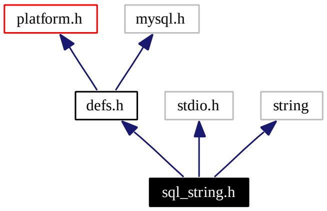
Declares an `std::string` derivative that adds some things needed within the library.

`#include "defs.h"`

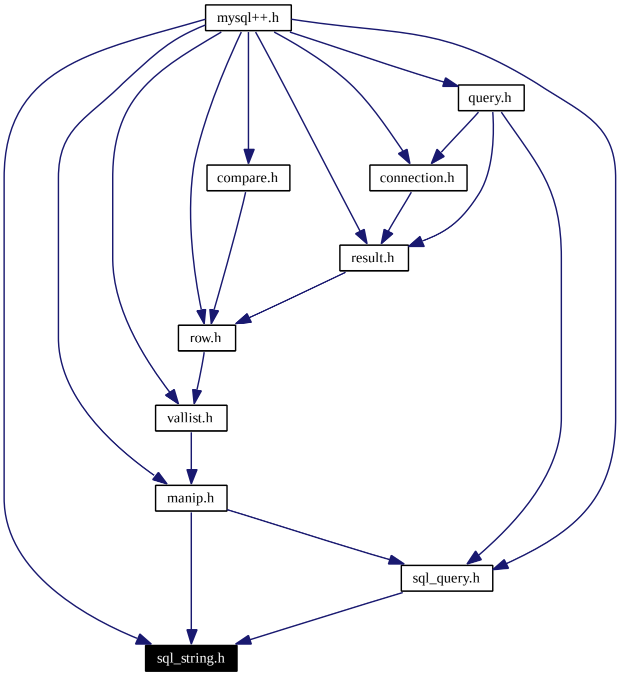
`#include <stdio.h>`

`#include <string>`

Include dependency graph for sql_string.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.23.1 Detailed Description

Declares an `std::string` derivative that adds some things needed within the library.

This class adds some flags needed by other parts of MySQL++, and it adds conversion functions from any primitive type. This helps in inserting these primitive types into the database, because we need everything in string form to build SQL queries.

8.24 sqlplus.hh File Reference

Deprecated backwards-compatibility header. Use **mysql++.h** in new code instead.

`#include "mysql++.h"`

Include dependency graph for sqlplus.hh:

8.24.1 Detailed Description

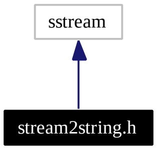
Deprecated backwards-compatibility header. Use **mysql++.h** in new code instead.

8.25 stream2string.h File Reference

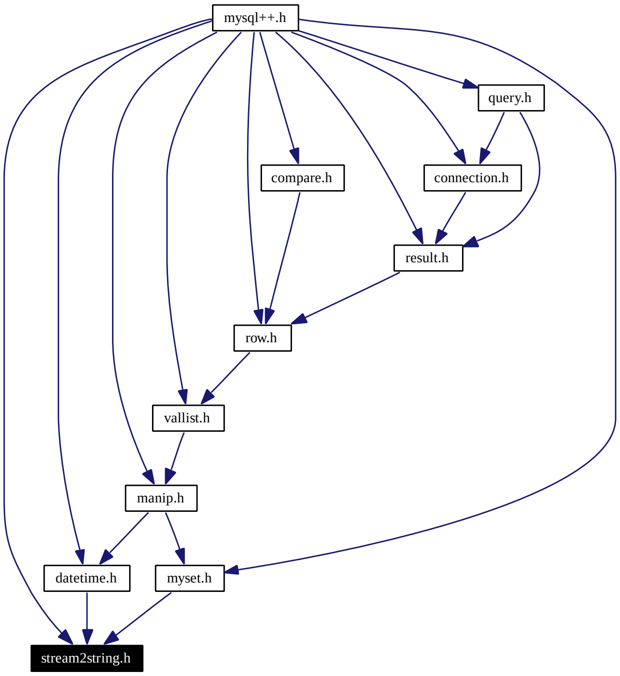
Declares an adapter that converts something that can be inserted into a C++ stream into a string type.

`#include <sstream>`

Include dependency graph for stream2string.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.25.1 Detailed Description

Declares an adapter that converts something that can be inserted into a C++ stream into a string type.

8.26 string_util.h File Reference

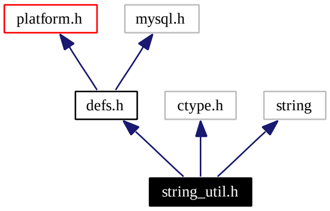
Declares string-handling utility functions used within the library.

`#include "defs.h"`

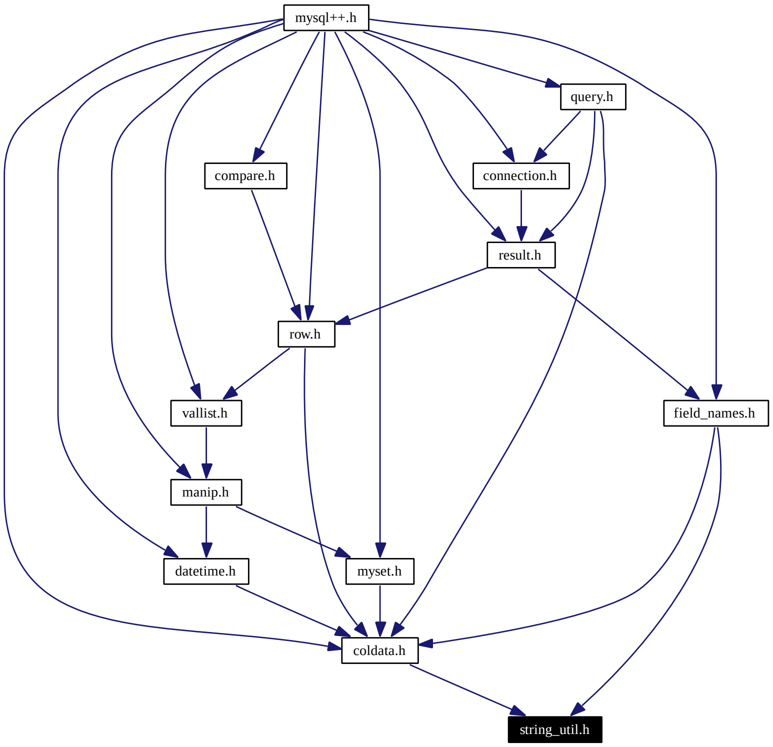
`#include <ctype.h>`

`#include <string>`

Include dependency graph for string_util.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

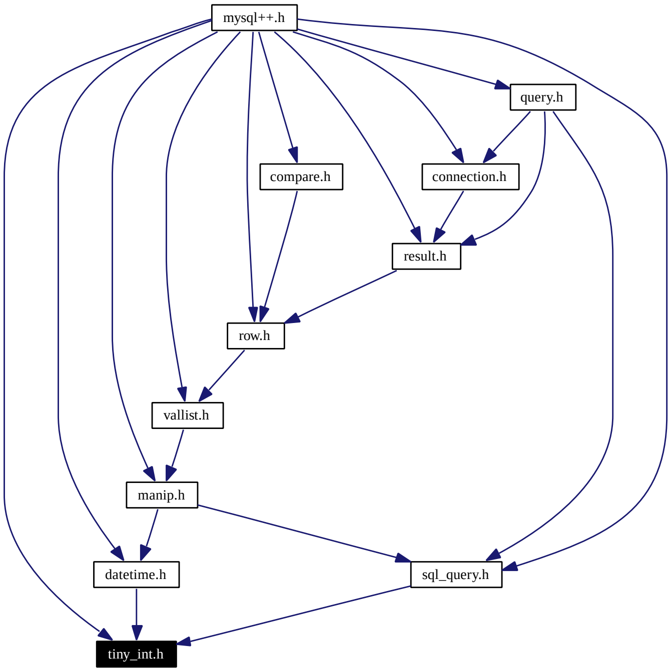
8.26.1 Detailed Description

Declares string-handling utility functions used within the library.

8.27 tiny_int.h File Reference

Declares class for holding a SQL tiny_int.

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.27.1 Detailed Description

Declares class for holding a SQL tiny_int.

8.28 type_info.h File Reference

Declares classes that provide an interface between the SQL and C++ type systems.

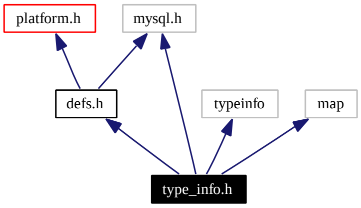
`#include "defs.h"`

`#include <mysql.h>`

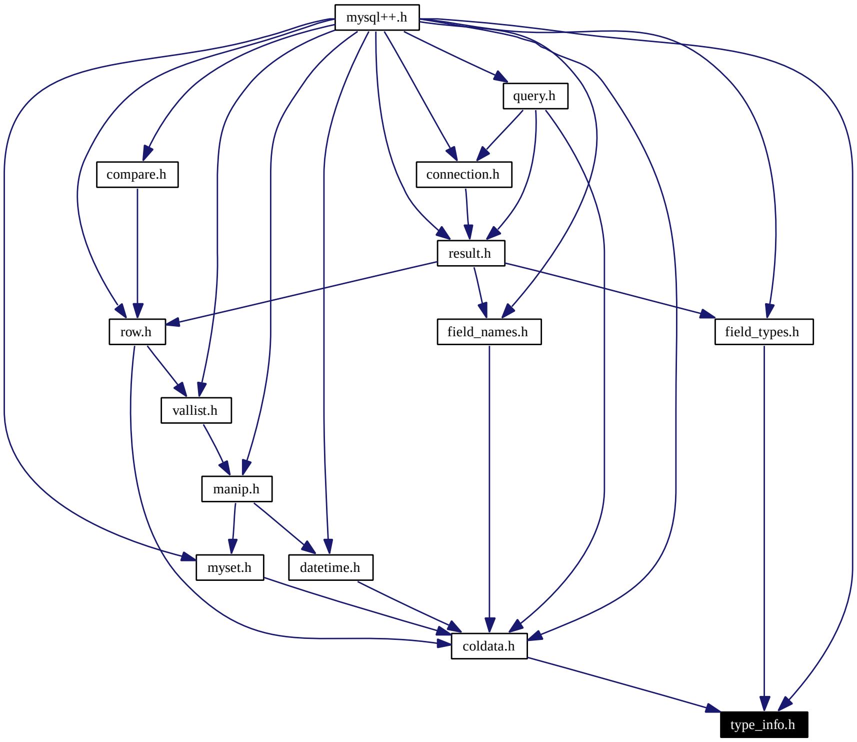
`#include <typeinfo>`

`#include <map>`

Include dependency graph for type_info.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.28.1 Detailed Description

Declares classes that provide an interface between the SQL and C++ type systems.

These classes are mostly used internal to the library.

8.29 vallist.h File Reference

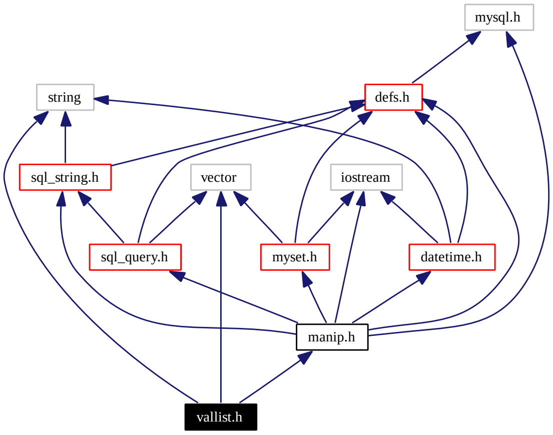
Declares templates for holding lists of values.

`#include "manip.h"`

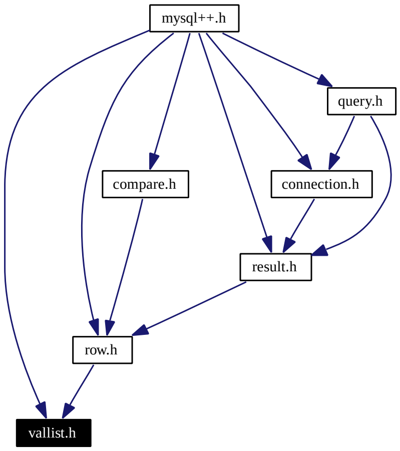
`#include <string>`

`#include <vector>`

Include dependency graph for vallist.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **mysqlpp**

8.29.1 Detailed Description

Declares templates for holding lists of values.

Index

base_type
    mysqlpp::mysql_type_info, 52
before
    mysqlpp::mysql_type_info, 52

c_type
    mysqlpp::mysql_type_info, 53
clear
    mysqlpp::SQLQueryParms, 81
close
    mysqlpp::Connection, 26
ColData
    mysqlpp, 14
coldata.h, 91
    operator_binary, 93
    operator_binary_int, 93
    oprsw, 93
compare.h, 94
connect
    mysqlpp::Connection, 26
connected
    mysqlpp::Connection, 26
Connection
    mysqlpp::Connection, 25
connection.h, 95
const_string.h, 97
convert
    mysqlpp::DateTime, 37
    mysqlpp::mysql_date, 47
    mysqlpp::mysql_time, 50
convert.h, 99
    mysql__convert, 100
copy
    mysqlpp::ResUse, 71

data_seek
    mysqlpp::Result, 67
datetime.h, 101
def
    mysqlpp::SQLQuery, 79
defs.h, 103
do_nothing
    mysqlpp, 16
do_nothing_type0
    mysqlpp, 16

eof
    mysqlpp::ResUse, 69
error
    mysqlpp::BadQuery, 22
    mysqlpp::Connection, 26
    mysqlpp::Query, 61
escape_q
    mysqlpp::ColData_Tmpl, 23
    mysqlpp::mysql_type_info, 53
escape_string
    mysqlpp, 15
escape_type0
    mysqlpp, 16
exceptions.h, 105
exec
    mysqlpp::Query, 62
execute
    mysqlpp::Query, 63

fetch_field
    mysqlpp::ResUse, 69
fetch_lengths
    mysqlpp::ResUse, 69
fetch_row
    mysqlpp::Result, 68
    mysqlpp::ResUse, 71
field_name
    mysqlpp::ResUse, 71
field_names
    mysqlpp::ResUse, 70
field_names.h, 107
field_num
    mysqlpp::ResUse, 71
field_seek
    mysqlpp::ResUse, 69
field_type
    mysqlpp::ResUse, 70
field_types
    mysqlpp::ResUse, 70
field_types.h, 108
fields
    mysqlpp::ResUse, 70
fields.h, 109

id

    mysqlpp::mysql_type_info, 53
ignore
    mysqlpp, 16
ignore_type0
    mysqlpp, 16
info
    mysqlpp::Connection, 24
    mysqlpp::ResNSel, 66
insert
    mysqlpp::Query, 63
iterator
    mysqlpp::const_string, 28

manip.h, 110
MutableColData
    mysqlpp, 14
myset.h, 112
mysql++.h, 114
mysql++.hh, 116
mysql__convert
    convert.h, 100
mysql_cmp
    mysqlpp, 17
mysql_cmp_cstr
    mysqlpp, 17
mysql_query_define0
    sql_query.h, 127
mysql_query_define1
    query.h, 121
mysql_query_define2
    query.h, 121
mysqlpp, 11
    ColData, 14
    do_nothing, 16
    do_nothing_type0, 16
    escape_string, 15
    escape_type0, 16
    ignore, 16
    ignore_type0, 16
    MutableColData, 14
    mysql_cmp, 17
    mysql_cmp_cstr, 17
    operator<<, 17
    quote, 17
    quote_double_only, 16
    quote_double_only_type0, 16
    quote_only, 16
    quote_only_type0, 16
    quote_type0, 16
    str_to_lwr, 15
    str_to_upr, 15
    stream2string, 17
    strip, 15
    strip_all_blanks, 15
    strip_all_non_num, 15
mysqlpp::BadConversion, 19
mysqlpp::BadFieldName, 20
mysqlpp::BadNullConversion, 21
mysqlpp::BadQuery, 22
    error, 22
mysqlpp::ColData_Tmpl, 23
    escape_q, 23
    quote_q, 23
    type, 23
mysqlpp::Connection, 24
    close, 26
    connect, 26
    connected, 26
    Connection, 25
    error, 26
    info, 24
    operator bool, 26
    query, 27
    real_connect, 27
    success, 27
mysqlpp::const_string, 28
    iterator, 28
mysqlpp::const_subscript_container, 29
mysqlpp::cstr_equal_to, 30
mysqlpp::cstr_greater, 31
mysqlpp::cstr_greater_equal, 32
mysqlpp::cstr_less, 33
mysqlpp::cstr_less_equal, 34
mysqlpp::cstr_not_equal_to, 35
mysqlpp::Date, 36
mysqlpp::DateTime, 37
    convert, 37
mysqlpp::DateTime
    out_stream, 37
mysqlpp::equal_list_b, 39
mysqlpp::equal_list_ba, 40
mysqlpp::escape_type1, 41
mysqlpp::escape_type2, 42
mysqlpp::FieldNames, 43
    operator=, 43
    operator[], 43
mysqlpp::Fields, 44
    operator[], 44
    size, 44
mysqlpp::FieldTypes, 45
    operator=, 45
    operator[], 45
mysqlpp::FieldTypes
    operator=, 45
mysqlpp::ignore_type2, 46
mysqlpp::mysql_date, 47
    convert, 47
    out_stream, 47

mysqlpp::mysql_dt_base, 49
mysqlpp::mysql_time, 50
    convert, 50
    out_stream, 50
mysqlpp::mysql_type_info, 52
    base_type, 52
    before, 52
    c_type, 53
    escape_q, 53
    id, 53
    name, 53
    quote_q, 53
    sql_name, 53
mysqlpp::MysqlCmp, 54
mysqlpp::MysqlCmpCStr, 55
    operator(), 55
mysqlpp::Null, 56
    Null, 56
mysqlpp::null_type, 57
mysqlpp::NullisBlank, 58
mysqlpp::NullisNull, 59
mysqlpp::NullisZero, 60
mysqlpp::Query, 61
    error, 61
    exec, 62
    execute, 63
    insert, 63
    preview, 61
    Query, 61, 62
    replace, 63
    store, 64
    storein, 64
    storein_sequence, 64
    storein_set, 65
    success, 61
    update, 65
    use, 65
mysqlpp::ResNSel, 66
    info, 66
    operator bool, 66
    rows, 66
mysqlpp::Result, 67
    data_seek, 67
    fetch_row, 68
    num_rows, 67
    operator[], 67
    rows, 67
    size, 67
mysqlpp::ResUse, 69
    eof, 69
    fetch_field, 69
    fetch_lengths, 69
    field_names, 70
    field_seek, 69
    field_type, 70
    field_types, 70
    fields, 70
    names, 70
    num_fields, 69
    reset_field_names, 70
    reset_field_types, 70
    reset_names, 70
    reset_types, 70
    table, 69
    types, 70
mysqlpp::ResUse
    copy, 71
    fetch_row, 71
    field_name, 71
    field_num, 71
mysqlpp::Row, 72
    operator bool, 72
    operator[], 72
    parent, 72
    self, 72
    size, 72
mysqlpp::RowTemplate, 74
mysqlpp::Set, 75
mysqlpp::simp_list_b, 76
mysqlpp::SQLParseElement, 77
mysqlpp::SQLQuery, 78
    def, 79
    proc, 78
    reset, 79
mysqlpp::SQLQueryNEParms, 80
mysqlpp::SQLQueryParms, 81
    clear, 81
    operator+=, 81
    operator<<, 81
    operator[], 81
mysqlpp::SQLQueryParms
    set, 81
mysqlpp::SQLString, 83
mysqlpp::subscript_iterator, 84
mysqlpp::Time, 85
mysqlpp::tiny_int, 86
mysqlpp::type_info_cmp, 87
mysqlpp::value_list_b, 88
mysqlpp::value_list_ba, 89
MYSQLPP_DLLEXPORT
    platform.h, 119

name
    mysqlpp::mysql_type_info, 53
names
    mysqlpp::ResUse, 70
Null
    mysqlpp::Null, 56

null.h, 117
num_fields
  mysqlpp::ResUse, 69
num_rows
  mysqlpp::Result, 67

operator bool
  mysqlpp::Connection, 26
  mysqlpp::ResNSel, 66
  mysqlpp::Row, 72
operator()
  mysqlpp::MysqlCmpCStr, 55
operator+=
  mysqlpp::SQLQueryParms, 81
operator<<
  mysqlpp, 17
  mysqlpp::SQLQueryParms, 81
operator=
  mysqlpp::FieldNames, 43
  mysqlpp::FieldTypes, 45
  mysqlpp::FieldTypes, 45
operator[]
  mysqlpp::FieldNames, 43
  mysqlpp::Fields, 44
  mysqlpp::FieldTypes, 45
  mysqlpp::Result, 67
  mysqlpp::Row, 72
  mysqlpp::SQLQueryParms, 81
operator_binary
  coldata.h, 93
operator_binary_int
  coldata.h, 93
oprsw
  coldata.h, 93
out_stream
  mysqlpp::DateTime, 37
  mysqlpp::mysql_date, 47
  mysqlpp::mysql_time, 50

parent
  mysqlpp::Row, 72
platform.h, 119
  MYSQLPP_DLLEXPORT, 119
preview
  mysqlpp::Query, 61
proc
  mysqlpp::SQLQuery, 78

Query
  mysqlpp::Query, 61, 62
query
  mysqlpp::Connection, 27
query.h, 120
  mysql_query_define1, 121
  mysql_query_define2, 121
quote
  mysqlpp, 17
quote_double_only
  mysqlpp, 16
quote_double_only_type0
  mysqlpp, 16
quote_only
  mysqlpp, 16
quote_only_type0
  mysqlpp, 16
quote_q
  mysqlpp::ColData_Tmpl, 23
  mysqlpp::mysql_type_info, 53
quote_type0
  mysqlpp, 16

real_connect
  mysqlpp::Connection, 27
replace
  mysqlpp::Query, 63
reset
  mysqlpp::SQLQuery, 79
reset_field_names
  mysqlpp::ResUse, 70
reset_field_types
  mysqlpp::ResUse, 70
reset_names
  mysqlpp::ResUse, 70
reset_types
  mysqlpp::ResUse, 70
resiter.h, 122
result.h, 123
row.h, 125
rows
  mysqlpp::ResNSel, 66
  mysqlpp::Result, 67

self
  mysqlpp::Row, 72
set
  mysqlpp::SQLQueryParms, 81
size
  mysqlpp::Fields, 44
  mysqlpp::Result, 67
  mysqlpp::Row, 72
sql_name
  mysqlpp::mysql_type_info, 53
sql_query.h, 126
  mysql_query_define0, 127
sql_string.h, 128
sqlplus.hh, 130
store
  mysqlpp::Query, 64

storein
    mysqlpp::Query, 64
storein_sequence
    mysqlpp::Query, 64
storein_set
    mysqlpp::Query, 65
str_to_lwr
    mysqlpp, 15
str_to_upr
    mysqlpp, 15
stream2string
    mysqlpp, 17
stream2string.h, 131
string_util.h, 132
strip
    mysqlpp, 15
strip_all_blanks
    mysqlpp, 15
strip_all_non_num
    mysqlpp, 15
success
    mysqlpp::Connection, 27
    mysqlpp::Query, 61

table
    mysqlpp::ResUse, 69
tiny_int.h, 134
type
    mysqlpp::ColData_Tmpl, 23
type_info.h, 135
types
    mysqlpp::ResUse, 70

update
    mysqlpp::Query, 65
use
    mysqlpp::Query, 65

vallist.h, 137