KrossKexiDB Reference Manual

1.0.

Generated by Doxygen 1.4.7

Fri Aug 4 03:04:11 2006

Contents

1 KrossKexiDB Namespace Index

1.1 KrossKexiDB Namespace List

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

2 KrossKexiDB Hierarchical Index

2.1 KrossKexiDB Class Hierarchy

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

3 KrossKexiDB Class Index

3.1 KrossKexiDB Class List

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

4 KrossKexiDB Namespace Documentation

4.1 Kross::KexiDB Namespace Reference

4.1.1 Detailed Description

Wrapper around the Kexi::KexiDB classes.

Only *Kross::Api* is used. So, this wrapper is independend to the used scripting backend and any interpreter who wraps the Kross-Api should be able to make such wrappers accessible from within scripting-languages.

Classes

- class KexiDBConnection
- class KexiDBConnectionData

- class KexiDBCursor
- class KexiDBDriver
- class KexiDBDriverManager
- class KexiDBField
- class KexiDBFieldList
- class KexiDBModule
- class KexiDBParser
- class KexiDBSchema
- class KexiDBTableSchema
- class KexiDBQuerySchema
- class KexiDBTransaction

5 KrossKexiDB Class Documentation

5.1 KexiDBConnection Class Reference

Inherits Class.

5.1.1 Detailed Description

A connection to a database.

Example (in Python) ;

```
# Import the kexidb module.
import krosskexidb
# Get the drivermanager.
drivermanager = krosskexidb.DriverManager()
# We need a connectiondata object.
connectiondata = drivermanager.createConnectionData()
# Fill the new connectiondata object with what we need to connect.
connectiondata.setFileName("/home/user/kexisqlite3file.kexi")
# Create the database-driver to access the SQLite3 backend.
driver = drivermanager.driver("SQLite3")
# Create the connection now.
connection = driver.createConnection(connectiondata)
# Establish the connection.
if not connection.connect(): raise("Failed to connect with db")
# Open database for usage. The filebased driver uses the filename as databasename.
if not connection.useDatabase("/home/user/kexisqlite3file.kexi"): raise("Failed to use db")
```

Private Member Functions

- bool hadError () const
- const QString lastError () const
- KexiDBConnectionData * data ()
- KexiDBDriver * driver ()
- bool connect ()
- bool isConnected ()
- bool disconnect ()
- bool isReadOnly () const
- bool databaseExists (const QString &dbname)
- const QString currentDatabase () const

- const QStringList databaseNames () const
- bool isDatabaseUsed () const
- bool useDatabase (const QString &dbname)
- bool closeDatabase ()
- const QStringList allTableNames () const
- const QStringList tableNames () const
- const QStringList queryNames () const
- KexiDBCursor * executeQueryString (const QString &sqlquery)
- KexiDBCursor * executeQuerySchema (KexiDBQuerySchema *queryschema)
- Kross::Api::Object::Ptr insertRecord (Kross::Api::List::Ptr)
- bool createDatabase (const QString &dbname)
- bool dropDatabase (const QString &dbname)
- bool createTable (KexiDBTableSchema *tableschema)
- bool dropTable (const QString &tablename)
- bool alterTable (KexiDBTableSchema *fromschema, KexiDBTableSchema *toschema)
- bool alterTableName (KexiDBTableSchema *tableschema, const QString &newtablename)
- KexiDBTableSchema * tableSchema (const QString &tablename) const
- bool isEmptyTable (KexiDBTableSchema *tableschema) const
- KexiDBQuerySchema * querySchema (const QString &queryname) const
- bool autoCommit () const
- bool setAutoCommit (bool enabled)
- KexiDBTransaction * beginTransaction ()
- bool commitTransaction (KexiDBTransaction *transaction)
- bool rollbackTransaction (KexiDBTransaction *transaction)
- KexiDBTransaction * defaultTransaction ()
- void setDefaultTransaction (KexiDBTransaction *transaction)
- Kross::Api::List * transactions ()
- KexiDBParser * parser ()
- void initialize ()

  Initialize the class instance.

5.1.2 Member Function Documentation

5.1.2.1 bool hadError () const [private]

Return true if there was an error during last operation on the database.

5.1.2.2 const QString lastError () const [private]

Return the last errormessage.

5.1.2.3 KexiDBConnectionData * data () [private]

Return the KexiDBConnectionData object used to create this connection.

5.1.2.4 KexiDBDriver * driver () [private]

Return the KexiDBDriver object this connection belongs too.

5.1.2.5 bool connect () [private]

Try to connect and return true if we are successfully connected now.

5.1.2.6 bool isConnected () [private]

Return true if we are connected.

5.1.2.7 bool disconnect () [private]

Disconnect and return true if we are successfully disconnected now.

5.1.2.8 bool isReadOnly () const [private]

Returns true if the connection is read-only.

5.1.2.9 bool databaseExists (const QString & *dbname*) [private]

Return true if the as argument passed databasename exists.

5.1.2.10 const QString currentDatabase () const [private]

Return the name of currently used database for this connection or empty string if there is no used database.

5.1.2.11 const QStringList databaseNames () const [private]

Return list of database names for opened connection.

5.1.2.12 bool isDatabaseUsed () const [private]

Return true if connection is properly established.

5.1.2.13 bool useDatabase (const QString & *dbname*) [private]

Opens an existing database specified by the as argument passed databasename and returns true if the database is used now.

5.1.2.14 bool closeDatabase () [private]

Closes currently used database for this connection.

5.1.2.15 const QStringList allTableNames () const [private]

Return names of all table schemas stored in currently used database include the internal KexiDB system table names (kexi__∗)

5.1.2.16 const QStringList tableNames () const [private]

Return names of all table schemas without the internal KexiDB system table names (kexi__∗)

5.1.2.17 const QStringList queryNames () const [private]

Return names of all query schemas stored in currently used database.

5.1.2.18 KexiDBCursor * executeQueryString (const QString & *sqlquery*) [private]

Executes query described by the as argument passed sqlstatement-string. Returns the opened cursor created for results of this query.

5.1.2.19 KexiDBCursor * executeQuerySchema (KexiDBQuerySchema * *queryschema*) [private]

Executes query described by the as argument passed KexiDBQuerySchema object. Returns the opened cursor created for results of this query.

5.1.2.20 Kross::Api::Object::Ptr insertRecord (Kross::Api::List::Ptr) [private]

Inserts the as argument passed KexiDBField object.

5.1.2.21 bool createDatabase (const QString & *dbname*) [private]

Creates new database with the as argument passed databasename.

5.1.2.22 bool dropDatabase (const QString & *dbname*) [private]

Drops the as argument passed databasename.

5.1.2.23 bool createTable (KexiDBTableSchema * *tableschema*) [private]

Creates table defined by the as argument passed KexiTableSchema object.

5.1.2.24 bool dropTable (const QString & *tablename*) [private]

Drops table defined by the as argument passed KexiDBTableSchema object.

5.1.2.25 bool alterTable (KexiDBTableSchema * *fromschema*, KexiDBTableSchema * *toschema*) [private]

Alters the as first argument passed KexiDBTableSchema object using the as second argument passed KexiDBTableSchema.

5.1.2.26 bool alterTableName (KexiDBTableSchema * *tableschema*, const QString & *newtablename*) [private]

Alters the tablename of the as first argument passed KexiDBTableSchema into the as second argument passed new tablename.

5.1.2.27 KexiDBTableSchema * tableSchema (const QString & *tablename*) const [private]

Returns the KexiDBTableSchema object of the table matching to the as argument passed tablename.

5.1.2.28 bool isEmptyTable (KexiDBTableSchema * *tableschema*) const [private]

Returns true if there is at least one valid record in the as argument passed tablename.

5.1.2.29 KexiDBQuerySchema * querySchema (const QString & *queryname*) const [private]

Returns the KexiDBQuerySchema object of the query matching to the as argument passed queryname.

5.1.2.30 bool autoCommit () const [private]

Return true if the \"auto commit\" option is on.

5.1.2.31 bool setAutoCommit (bool *enabled*) [private]

Set the auto commit option. This does not affect currently started transactions and can be changed even when connection is not established.

5.1.2.32 KexiDBTransaction * beginTransaction () [private]

Creates new transaction handle and starts a new transaction.

5.1.2.33 bool commitTransaction (KexiDBTransaction * *transaction*) [private]

Commits the as rgument passed KexiDBTransaction object.

5.1.2.34 bool rollbackTransaction (KexiDBTransaction * *transaction*) [private]

Rollback the as rgument passed KexiDBTransaction object.

5.1.2.35 KexiDBTransaction * defaultTransaction () [private]

Return the KEXIDBTransaction object for default transaction for this connection.

5.1.2.36 void setDefaultTransaction (KexiDBTransaction * *transaction*) [private]

Sets default transaction that will be used as context for operations on data in opened database for this connection.

5.1.2.37 Kross::Api::List * transactions () [private]

Return list of currently active KexiDBTransaction objects.

5.1.2.38 KexiDBParser * parser () [private]

Return a KexiDBParser object.

5.2 KexiDBConnectionData Class Reference

Inherits Class.

5.2.1 Detailed Description

A KexiDBConnectionData is used to store the details needed for a connection with a database.

Private Member Functions

- const QString caption () const
- void setCaption (const QString &name)
- const QString description () const
- void setDescription (const QString &desc)
- const QString driverName () const
- void setDriverName (const QString &driver)
- bool localSocketFileUsed () const
- void setLocalSocketFileUsed (bool used)
- const QString localSocketFileName () const
- void setLocalSocketFileName (const QString &socketfilename)
- const QString databaseName () const
- void setDatabaseName (const QString &dbname)
- const QString hostName () const
- void setHostName (const QString &hostname)
- int port () const
- void setPort (int p)
- const QString password () const
- void setPassword (const QString &passwd)
- const QString userName () const
- void setUserName (const QString &username)
- const QString fileName () const
- void setFileName (const QString &filename)
- const QString dbPath () const
- const QString dbFileName () const
- const QString serverInfoString () const

5.2.2 Member Function Documentation

5.2.2.1 const QString caption () const [private]

Return the connection name.

5.2.2.2 void setCaption (const QString & *name*) [private]

Set the connection name.

5.2.2.3 const QString description () const [private]

Return the description.

5.2.2.4 void setDescription (const QString & *desc*) [private]

Set the description.

5.2.2.5 const QString driverName () const [private]

Return drivername.

5.2.2.6 void setDriverName (const QString & *driver*) [private]

Set the drivername.

5.2.2.7 bool localSocketFileUsed () const [private]

Return true if a local socket file is used else false.

5.2.2.8 void setLocalSocketFileUsed (bool *used*) [private]

Set if the local socket file should be used.

5.2.2.9 const QString localSocketFileName () const [private]

Return the local socket filename.

5.2.2.10 void setLocalSocketFileName (const QString & *socketfilename*) [private]

Set the local socket filename.

5.2.2.11 const QString databaseName () const [private]

Return the database name.

5.2.2.12 void setDatabaseName (const QString & *dbname*) [private]

Set the database name.

5.2.2.13 const QString hostName () const [private]

Return the hostname.

5.2.2.14 void setHostName (const QString & *hostname*) [private]

Set the hostname.

5.2.2.15 int port () const [private]

Return the port number.

5.2.2.16 void setPort (int *p*) [private]

Set the port number.

5.2.2.17 const QString password () const [private]

Return the password.

5.2.2.18 void setPassword (const QString & *passwd*) [private]

Set the password.

5.2.2.19 const QString userName () const [private]

Return the username.

5.2.2.20 void setUserName (const QString & *username*) [private]

Set the username.

5.2.2.21 const QString fileName () const [private]

Return the filename.

5.2.2.22 void setFileName (const QString & *filename*) [private]

Set the filename.

5.2.2.23 const QString dbPath () const [private]

Return the database path.

5.2.2.24 const QString dbFileName () const [private]

Return the database filename.

5.2.2.25 const QString serverInfoString () const [private]

Return a user-friendly string representation.

5.3 KexiDBCursor Class Reference

Inherits Class.

5.3.1 Detailed Description

The cursor provides a control structure for the successive traversal of records in a result set as returned e.g. by a query.

Example (in Python) that shows how to iterate over the result of a query;

```
# Once we have a KexiDBConnection object we are able to execute a query string and get a cursor as result
cursor = connection.executeQueryString("SELECT * from emp")
# Let's check if the query was successfully.
if not cursor: raise("Query failed")
# Walk through all items in the table.
while(not cursor.eof()):
    # Iterate over the fields the record has.
    for i in range( cursor.fieldCount() ):
        # Print some informations.
        print "%s %s %s" % (cursor.at(), i, cursor.value(i))
    # and move on to the next record.
    cursor.moveNext()
```

Example (in Python) that shows how to use a cursor to strip all whitespaces at the beginning and the end from the values in a table;

```
import krosskexidb
drivermanager = krosskexidb.DriverManager()
connectiondata = drivermanager.createConnectionDataByFile("/home/me/kexiprojectfile.kexi")
driver = drivermanager.driver( connectiondata.driverName() )
connection = driver.createConnection(connectiondata)
if not connection.connect(): raise "Failed to connect"
if not connection.useDatabase( connectiondata.databaseName() ):
    if not connection.useDatabase( connectiondata.fileName() ):
        raise "Failed to use database"

table = connection.tableSchema("emp")
query = table.query()
cursor = connection.executeQuerySchema(query)
if not cursor: raise("Query failed")
while(not cursor.eof()):
    for i in range( cursor.fieldCount() ):
        v = str( cursor.value(i) )
        if v.startswith(' ') or v.endswith(' '):
            cursor.setValue(i, v.strip())
    cursor.moveNext()
if not cursor.save(): raise "Failed to save changes"
```

Private Member Functions

- bool open ()
- bool isOpened ()
- bool reopen ()
- bool close ()
- bool moveFirst ()
- bool moveLast ()
- bool movePrev ()
- bool moveNext ()
- bool bof ()
- bool eof ()
- Q_LLONG at ()
- uint fieldCount ()
- QVariant value (uint index)
- bool setValue (uint index, QVariant value)
- bool save ()

5.3.2 Member Function Documentation

5.3.2.1 bool open () [private]

Opens the cursor.

5.3.2.2 bool isOpened () [private]

Returns true if the cursor is opened else false.

5.3.2.3 bool reopen () [private]

Closes and then opens again the same cursor.

5.3.2.4 bool close () [private]

Closes previously opened cursor.

5.3.2.5 bool moveFirst () [private]

Moves current position to the first record and retrieves it.

5.3.2.6 bool moveLast () [private]

Moves current position to the last record and retrieves it.

5.3.2.7 bool movePrev () [private]

Moves current position to the previous record and retrieves it.

5.3.2.8 bool moveNext () [private]

Moves current position to the next record and retrieves it.

5.3.2.9 bool bof () [private]

Returns true if current position is before first record.

5.3.2.10 bool eof () [private]

Returns true if current position is after last record.

5.3.2.11 Q_LLONG at () [private]

Returns current internal position of the cursor's query. Records are numbered from 0; the value -1 means that the cursor does not point to a valid record.

5.3.2.12 uint fieldCount () [private]

Returns the number of fields available for this cursor.

5.3.2.13 QVariant value (uint *index*) [private]

Returns the value stored in the passed column number (counting from 0).

5.3.2.14 bool setValue (uint *index*, QVariant *value*) [private]

Set the value for the field defined with index. The new value is buffered and does not got written as long as save() is not called.

5.3.2.15 bool save () [private]

Save any changes done with setValue(). You should call this only once at the end of all value/setValue iterations cause the cursor is closed once the changes got saved successfully.

5.4 KexiDBDriver Class Reference

Inherits Class.

5.4.1 Detailed Description

Drivers are the implementations Kexi uses to access the driver-backends.

Example (in Python) ;

```
# Import the kexidb module.
import krosskexidb
# Get the drivermanager.
drivermanager = krosskexidb.DriverManager()
# Create the driver now.
driver = drivermanager.driver("SQLite3")
# Check if the driver is valid.
if not driver.isValid(): raise "Invalid driver"
# Create a connectiondata object.
connectiondata = drivermanager.createConnectionData()
# Fill the new connectiondata object with what we need to connect.
connectiondata.setFileName("/home/user/kexisqlite3file.kexi")
# Print the list of connections before.
print driver.connectionsList()
# Create the connection now.
connection = driver.createConnection(connectiondata)
# Print the list of connections again. This includes our just created connection now.
print driver.connectionsList()
```

Private Member Functions

- bool isValid ()
- int versionMajor ()
- int versionMinor ()
- QString escapeString (const QString &s)
- bool isFileDriver ()
- QString fileDBDriverMimeType ()
- bool isSystemObjectName (const QString &name)
- bool isSystemDatabaseName (const QString &name)
- bool isSystemFieldName (const QString &name)
- QString valueToSQL (const QString &fieldtype, const QVariant &value)
- KexiDBConnection * createConnection (KexiDBConnectionData *data)
- QPtrList< ::KexiDB::Connection > connectionsList ()

5.4.2 Member Function Documentation

5.4.2.1 bool isValid () [private]

Return true if this driver is valid else false.

5.4.2.2 int versionMajor () [private]

The drivers major versionnumber.

5.4.2.3 int versionMinor () [private]

The drivers minor versionnumber.

5.4.2.4 QString escapeString (const QString & *s*) [private]

Driver-specific SQL string escaping. For example the " or ' char may need to be escaped for values used within SQL-statements.

5.4.2.5 bool isFileDriver () [private]

Returns true if this driver is file-based.

5.4.2.6 QString fileDBDriverMimeType () [private]

Return a name of MIME type of files handled by this driver if it is a file-based database's driver otherwise returns null string.

5.4.2.7 bool isSystemObjectName (const QString & *name*) [private]

Returns true if the passed string is a system object's name, eg. name of build-in system table that cannot be used or created by a user.

5.4.2.8 bool isSystemDatabaseName (const QString & *name*) [private]

Returns true if the passed string is a system database's name, eg. name of build-in, system database that cannot be used or created by a user.

5.4.2.9 bool isSystemFieldName (const QString & *name*) [private]

Returns true if the passed string is a system field's name, build-in system field that cannot be used or created by a user.

5.4.2.10 QString valueToSQL (const QString & *fieldtype*, const QVariant & *value*) [private]

The as second argument passed string got escaped to be usuable within a SQL-statement and those escaped string got returned by the method. The first argument defines the fieldtype to what we should escape the second argument to.

5.4.2.11 KexiDBConnection * createConnection (KexiDBConnectionData * *data*) [private]

Create a new KexiDBConnection object and return it.

5.4.2.12 QPtrList<::KexiDB::Connection > connectionsList () [private]

Return a list of KexiDBConnection objects.

5.5 KexiDBDriverManager Class Reference

Inherits Class.

5.5.1 Detailed Description

The drivermanager is the base class to access KexiDBDriver objects and provides common functionality to deal with the KexiDB module.

Example (in Python) ;

```
# Import the kexidb module.
import krosskexidb
# Get the drivermanager.
drivermanager = krosskexidb.DriverManager()
# Let's determinate the mimetype (e.g. "application/x-sqlite3").
mimetype = drivermanager.mimeForFile("/home/user/mykexidbfile.kexi")
# Now we use that mimetype to get the name of the driver to handle that file (e.g. "SQLite3")
drivername = drivermanager.lookupByMime(mimetype)
# We are able to create the driver now.
driver = drivermanager.driver(drivername)
```

Private Member Functions

- const QStringList driverNames ()
- KexiDBDriver * driver (const QString &drivername)
- const QString lookupByMime (const QString &mimetype)
- const QString mimeForFile (const QString &filename)
- KexiDBConnectionData * createConnectionData ()
- KexiDBConnectionData * createConnectionDataByFile (const QString &filename)
- KexiDBField * field ()
- KexiDBTableSchema * tableSchema (const QString &tablename)
- KexiDBQuerySchema * querySchema ()

5.5.2 Member Function Documentation

5.5.2.1 const QStringList driverNames () [private]

Returns a list with avaible drivernames.

5.5.2.2 KexiDBDriver * driver (const QString & *drivername*) [private]

Return the to the defined drivername matching KexiDBDriver object.

5.5.2.3 const QString lookupByMime (const QString & *mimetype*) [private]

Return the to the defined mimetype-string matching drivername.

5.5.2.4 const QString mimeForFile (const QString & *filename*) [private]

Return the matching mimetype for the defined file.

5.5.2.5 KexiDBConnectionData * createConnectionData () [private]

Return a new KexiDBConnectionData object.

5.5.2.6 KexiDBConnectionData * createConnectionDataByFile (const QString & *filename*) [private]

Create and return a KexiDBConnectionData object. Fill the content of the KexiDBConnectionData object with the defined file as. The file could be e.g. a *.kexi file or a *.kexis file.

5.5.2.7 KexiDBField * field () [private]

Return a new KexiDBField object.

5.5.2.8 KexiDBTableSchema * tableSchema (const QString & *tablename*) [private]

Return a new KexiDBTableSchema object.

5.5.2.9 KexiDBQuerySchema * querySchema () [private]

Return a new KexiDBQuerySchema object.

5.6 KexiDBField Class Reference

Inherits Class.

5.6.1 Detailed Description

A field in a record.

Private Member Functions

- const QString type ()
- void setType (const QString type)
- const QString subType ()
- void setSubType (const QString &subtype)
- const QString variantType ()
- const QString typeGroup ()
- bool isAutoInc ()
- void setAutoInc (bool autoinc)
- bool isUniqueKey ()
- void setUniqueKey (bool unique)
- bool isPrimaryKey ()
- void setPrimaryKey (bool primary)
- bool isForeignKey ()
- void setForeignKey (bool foreign)
- bool isNotNull ()
- void setNotNull (bool notnull)
- bool isNotEmpty ()
- void setNotEmpty (bool notempty)
- bool isIndexed ()
- void setIndexed (bool indexed)
- bool isUnsigned ()
- void setUnsigned (bool isunsigned)

- const QString name ()
- void setName (const QString &name)
- const QString caption ()
- void setCaption (const QString &caption)
- const QString description ()
- void setDescription (const QString &desc)
- uint length ()
- void setLength (uint length)
- uint precision ()
- void setPrecision (uint precision)
- uint width ()
- void setWidth (uint width)
- QVariant defaultValue ()
- void setDefaultValue (const QVariant &defaultvalue)

5.6.2 Member Function Documentation

5.6.2.1 const QString type () [private]

Returns the type string for this field, e.g. "Integer" for Integer type.

5.6.2.2 void setType (const QString *type*) [private]

Sets the type string for this field, e.g. "Integer" for Integer type.

5.6.2.3 const QString subType () [private]

Returns the optional subtype for this field. Subtype is a string providing additional hint for field's type. E.g. for BLOB type, it can be a MIME type or certain QVariant type name, for example: "QPixmap", "QColor" or "QFont".

5.6.2.4 void setSubType (const QString & *subtype*) [private]

Sets the optional subtype for this field.

5.6.2.5 const QString variantType () [private]

Returns the QVariant::typeName which is equivalent to the type this field has.

5.6.2.6 const QString typeGroup () [private]

Returns type group string for this field, e.g. "IntegerGroup" for IntegerGroup type.

5.6.2.7 bool isAutoInc () [private]

Returns true if the field is autoincrement (e.g. integer/numeric).

5.6.2.8 void setAutoInc (bool *autoinc*) [private]

Sets auto increment flag.

5.6.2.9 bool isUniqueKey () [private]

Returns true if the field is member of single-field unique key.

5.6.2.10 void setUniqueKey (bool *unique*) [private]

Specifies whether the field has single-field unique constraint or not.

5.6.2.11 bool isPrimaryKey () [private]

Returns true if the field is member of single-field primary key.

5.6.2.12 void setPrimaryKey (bool *primary*) [private]

Specifies whether the field is single-field primary key or not.

5.6.2.13 bool isForeignKey () [private]

Returns true if the field is member of single-field foreign key.

5.6.2.14 void setForeignKey (bool *foreign*) [private]

Sets whether the field has to be declared with single-field foreign key.

5.6.2.15 bool isNotNull () [private]

Returns true if the field is not allowed to be null.

5.6.2.16 void setNotNull (bool *notnull*) [private]

Specifies whether the field has single-field unique constraint or not.

5.6.2.17 bool isNotEmpty () [private]

Returns true if the field is not allowed to be empty.

5.6.2.18 void setNotEmpty (bool *notempty*) [private]

Specifies whether the field has single-field unique constraint or not.

5.6.2.19 bool isIndexed () [private]

Returns true if the field is indexed using single-field database index.

5.6.2.20 void setIndexed (bool *indexed*) [private]

Specifies whether the field is indexed or not.

5.6.2.21 bool isUnsigned () [private]

Returns true if the field is an unsigned integer.

5.6.2.22 void setUnsigned (bool *isunsigned*) [private]

Specifies whether the field is an unsigned integer or not.

5.6.2.23 const QString name () [private]

Returns the name of this field.

5.6.2.24 void setName (const QString & *name*) [private]

Sets the name of this field.

5.6.2.25 const QString caption () [private]

Returns the caption of this field.

5.6.2.26 void setCaption (const QString & *caption*) [private]

Sets the caption of this field.

5.6.2.27 const QString description () [private]

Returns the descriptive text for this field.

5.6.2.28 void setDescription (const QString & *desc*) [private]

Set the description for this field.

5.6.2.29 uint length () [private]

Returns the length of text if the field type is text.

5.6.2.30 void setLength (uint *length*) [private]

Sets the length for this field. Only works for Text Type (not including LongText).

5.6.2.31 uint precision () [private]

Returns precision for numeric and other fields that have both length and precision (floating point types).

5.6.2.32 void setPrecision (uint *precision*) [private]

Sets the precision for numeric and other fields.

5.6.2.33 uint width () [private]

Returns the width of this field (usually in pixels or points). 0 (the default) means there is no hint for the width.

5.6.2.34 void setWidth (uint *width*) [private]

Sets the width of this field.

5.6.2.35 QVariant defaultValue () [private]

Returns the default value this field has.

5.6.2.36 void setDefaultValue (const QVariant & *defaultvalue*) [private]

Sets the default value this field has.

5.7 KexiDBFieldList Class Reference

Inherits Class.

5.7.1 Detailed Description

A list of fields. The KexiDBFieldList can be used to handle KexiDBField objects in a backend-independend way.

Example (in Python) ;

```
# Get the tableschema for the "dept" table.
table = connection.tableSchema("dept")
# Create a KexiDBFieldList based on the table and filled with the selected fields.
subfields = ["deptno","name","loc"]
fieldlist = table.fieldlist().subList(subfields)
# Create the "SELECT * from dept;" queryschema.
query = table.query()
# We change the queryschema to "SELECT deptno,name,loc FROM dept;" now.
query.fieldlist().setFields(fieldlist)
# and change the query to "SELECT deptno,name,loc FROM dept WHERE deptno=5;"
query.setWhereExpression("deptno=5")
# Execute the query and get a KexiDBCursor object as result which could be used to iterate through the re
cursor = connection.executeQuerySchema(query)
```

Private Member Functions

- uint fieldCount ()
- KexiDBField * field (uint index)
- KexiDBField * fieldByName (const QString &name)
- Kross::Api::List * fields ()
- bool hasField (KexiDBField *field)
- const QStringList names () const
- void addField (KexiDBField *field)
- void insertField (uint index, KexiDBField *field)
- void removeField (KexiDBField *field)
- void clear ()
- void setFields (KexiDBFieldList *fieldlist)
- KexiDBFieldList * subList (QValueList< QVariant > list)

5.7.2 Member Function Documentation

5.7.2.1 uint fieldCount () [private]

Returns the number of fields.

5.7.2.2 KexiDBField * field (uint *index*) [private]

Return the field specified by the index-number passed as an argument.

5.7.2.3 KexiDBField * fieldByName (const QString & *name*) [private]

Return the field specified by the as an argument passed fieldname.

5.7.2.4 Kross::Api::List * fields () [private]

Returns a list of all fields.

5.7.2.5 bool hasField (KexiDBField * *field*) [private]

Returns true if the KexiDBField object passed as an argument is in the field list.

5.7.2.6 const QStringList names () const [private]

Return a list of field names.

5.7.2.7 void addField (KexiDBField * *field*) [private]

Adds the KexiDBField object passed as an argument to the field list.

5.7.2.8 void insertField (uint *index*, KexiDBField * *field*) [private]

Inserts the KexiDBField object passed as the second argument into the field list at the position defined by the first argument.

5.7.2.9 void removeField (KexiDBField * *field*) [private]

Removes the KexiDBField object passed as an argument from the field list.

5.7.2.10 void clear () [private]

Removes all KexiDBField objects from the fieldlist.

5.7.2.11 void setFields (KexiDBFieldList * *fieldlist*) [private]

Set the fieldlist to the as argument passed list of fields.

5.7.2.12 KexiDBFieldList * subList (QValueList< QVariant > *list*) [private]

Creates and returns list that contain fields selected by name.

5.8 KexiDBModule Class Reference

Inherits Module.

5.8.1 Detailed Description

The KexiDBModule wrapper around KexiDB.

This class implementates a *Kross::Api::Module* to wrap those parts of KexiDB that should be accessible for the different scripting languages. All work will be done and presented in the common Kross::Api and therefore is independend of any used scripting-backend.

Public Member Functions

- virtual Kross::Api::Object::Ptr get (const QString &name, void *p=0)

5.8.2 Member Function Documentation

5.8.2.1 Kross::Api::Object::Ptr get (const QString & *name*, void * *p* = 0) [virtual]

Variable module-method use to call transparent some functionality the module provides.

Parameters:

name A name passed to the method. This name is used internaly to determinate what the caller likes to do. Each implemented module have to implement what should be done.

p A variable pointer passed to the method. It depends on the module and the name what this pointer is.

Returns:

a *Kross::Api::Object* or NULL.

5.9 KexiDBParser Class Reference

Inherits Class.

5.9.1 Detailed Description

The KexiDBParser could be used to parse SQL-statements.

Example (in Python) ;

```
# First we need a parser object.
parser = connection.parser()
# Parse a SQL-statement.
parser.parse("SELECT * from table1")
# The operation could be e.g. SELECT or INSERT.
if parser.operation() == 'Error':
    raise parser.errorMsg()
# Print some feedback.
print "Successfully parsed the SQL-statement %s" % parser.statement()
```

Private Member Functions

- bool parse (const QString &sql)
- void clear ()
- const QString operation ()

- KexiDBTableSchema * table ()
- KexiDBQuerySchema * query ()
- KexiDBConnection * connection ()
- const QString statement ()
- const QString errorType ()
- const QString errorMsg ()
- int errorAt ()

5.9.2 Member Function Documentation

5.9.2.1 bool parse (const QString & *sql*) [private]

Clears previous results and runs the parser on the SQL statement passed as an argument.

5.9.2.2 void clear () [private]

Clears parsing results.

5.9.2.3 const QString operation () [private]

Returns the resulting operation.

5.9.2.4 KexiDBTableSchema * table () [private]

Returns the KexiDBTableSchema object on a CREATE TABLE operation.

5.9.2.5 KexiDBQuerySchema * query () [private]

Returns the KexiDBQuerySchema object on a SELECT operation.

5.9.2.6 KexiDBConnection * connection () [private]

Returns the KexiDBConnection object pointing to the used database connection.

5.9.2.7 const QString statement () [private]

Returns the SQL query statement.

5.9.2.8 const QString errorType () [private]

Returns the type string of the last error.

5.9.2.9 const QString errorMsg () [private]

Returns the message of the last error.

5.9.2.10 int errorAt () [private]

Returns the position where the last error occurred.

5.10 KexiDBQuerySchema Class Reference

Inherits KexiDBSchema< Kross::KexiDB::KexiDBQuerySchema >.

5.10.1 Detailed Description

The KexiDBTableSchema object implements a KexiDBSchema for queries.

Private Member Functions

- const QString statement () const
- void setStatement (const QString &statement)
- bool setWhereExpression (const QString &whereexpression)

5.10.2 Member Function Documentation

5.10.2.1 const QString statement () const [private]

Returns the SQL-statement of this query schema.

5.10.2.2 void setStatement (const QString & *statement*) [private]

Set the SQL-statement of this query schema.

5.10.2.3 bool setWhereExpression (const QString & *whereexpression*) [private]

Set the where-expression.

5.11 KexiDBSchema Class Template Reference

Inherits Class.

Inherited by KexiDBQuerySchema, and KexiDBTableSchema.

5.11.1 Detailed Description

template<class T> class Kross::KexiDB::KexiDBSchema< T >

The KexiDBSchema object provides common functionality for schemas like KexiDBTableSchema or KexiDBQuerySchema.

Example (in Python) ;

```
# Get the tableschema from a KexiDBConnection object.
tableschema = connection.tableSchema("dept")
# Print some informations.
print "table=%s description=%s" % (tableschema.name(), tableschema.description())
# Get the "SELECT * FROM dept;" queryschema for the table.
queryschema = tableschema.query()
# Walk through the fields/columns the queryschema has and print the fieldnames.
for field in queryschema.fieldlist().fields():
    print "fieldname=%s" % field.name()
# Execute the query. The returned KexiDBCursor object could be used then to iterate through the result.
cursor = connection.executeQuerySchema(queryschema)
```

Private Member Functions

- const QString name () const
- void setName (const QString &name)
- const QString caption () const
- void setCaption (const QString &caption)
- const QString description () const
- void setDescription (const QString &description)
- KexiDBFieldList * fieldlist () const

5.11.2 Member Function Documentation

5.11.2.1 const QString name () const [private]

Returns the name of the schema.

5.11.2.2 void setName (const QString & *name*) [private]

Set the name of the schema.

5.11.2.3 const QString caption () const [private]

Returns the caption of the schema.

5.11.2.4 void setCaption (const QString & *caption*) [private]

Set the caption of the schema.

5.11.2.5 const QString description () const [private]

Returns a description of the schema.

5.11.2.6 void setDescription (const QString & *description*) [private]

Set a description of the schema.

5.11.2.7 KexiDBFieldList * fieldlist () const [private]

Returns the KexiDBFieldList object this schema has.

5.12 KexiDBTableSchema Class Reference

Inherits KexiDBSchema< Kross::KexiDB::KexiDBTableSchema >.

5.12.1 Detailed Description

The KexiDBTableSchema object implements a KexiDBSchema for tables.

Private Member Functions

- KexiDBQuerySchema * query ()

5.12.2 Member Function Documentation

5.12.2.1 KexiDBQuerySchema * query () [private]

Return the KexiDBQuerySchema object that represents a "SELECT * FROM this_KexiDBTableSchema_-object" SQL-statement.

5.13 KexiDBTransaction Class Reference

Inherits Class.

5.13.1 Detailed Description

Transactions are used to ensure that integrity of a database is maintained.

Private Member Functions

- bool isActive () const
- bool isNull () const

5.13.2 Member Function Documentation

5.13.2.1 bool isActive () const [private]

Return true if the transaction is active (ie. started).

5.13.2.2 bool isNull () const [private]

Return true if the transaction is uninitialized (null).

Index