Intentions good, warranty void

Using Qt in unexpected ways

Till Adam / Mirko Böhlm
In which MB holds forth and TA smiles and nods ...
git.kde.org/kdevplatform/interfaces/foregroundlock.h

Copyright 2010 David Nolden <david.nolden.kdevelop@art-master.de>

(you old devil, you)
The Problem

You: “I need to do something non-thread-safe from a secondary thread.”
Reasonable People: “Don't, the GUI thread could be doing anything.”
You: “But, but, I have to build the shrubbery! Can I?”
Reasonable People: “Just don't!”
You: “I am a programmer, I control the narrative, I will not be thwarted by convention or the fears of lesser mortals!”
Reasonable People: “Yeah, whatever, you really do have to get out more...”
The Solution

On the secondary thread:

- Signal we want to acquire the lock
- Schedule method invocation on the main thread via event loop (\texttt{QMetaObject::invokeMethod}, \texttt{Qt::QueuedConnection}),
- wait on waitcondition
The Solution continued

On the primary thread:

- Lock global mutex on static object
- Signal the lock is held by waking up requester
- Wait for requester to be done
- Return control to event loop
ForegroundLock
What Qt gives us today:

```cpp
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
    db.setHostName("bigblue");
    db.setDatabaseName("flightdb");
    db.setUserName("acarlson");
    db.setPassword("1uTbSbAs");
    bool ok = db.open();

QSqlQuery query;
    query.exec("SELECT name, salary FROM employee WHERE salary > 50000");

while (query.next()) {
    QString name = query.value(0).toString();
    int salary = query.value(1).toInt();
    qDebug() << name << salary;
}
```
We would like to have:

- automatable type-safe table creation
- safely and robustly generated queries
- type-safe select statements

=> strings are, like, totally 80s
Industrial Strength SQL

Type-safe table creation:

- Describe structure in domain specific language (*cough* macros *cough*)
- Use TMP to get compile time column type checking
- Generate QtSQL code to create tables
TABLE( Staff, SQLDRIVER_EXPORT ) {
    ONLY_USER_SELECT
    SQL_NAME( "tblStaff" );
    COLUMN( id, QUuid, PrimaryKey );
    FOREIGN_KEY( fk_lutTitle_id, Title, id, NotNull|OnDeleteRestrict );
    COLUMN( StaffForename, QString, Null, 128 );
    COLUMN( StaffMiddlename, QString, Null, 128 );
    COLUMN( StaffSurname, QString, Null, 128 );
    COLUMN( StaffRegNum, QString, Null, 32 );
    COLUMN( StaffSuffix, QString, Null, 128 );
    FOREIGN_KEY( StaffGrade, StaffGrades, id, NotNull|OnDeleteRestrict );
    COLUMN( AdmitRights, bool, Null );
    COLUMN( StaffActive, bool, Null );
    COLUMN( UserName, QString, Unique, 255 );

    typedef boost::mpl::vector<
idType, fk_lutTitle_idType, StaffForenameType,
    StaffMiddlenameType, StaffSurnameType, StaffRegNumType, StaffSuffixType,
    StaffGradeType, AdmitRightsTy
    >;
};
/** Convenience macro to create a table type. */
#define TABLE( name, _EXPORT )
    struct name ## Type; 
    extern _EXPORT name ## Type name; 
    struct name ## Type : Table< name ## Type >

template <typename T>
QString createTableStatement()
{
    QStringList cols;
    detail::column_creator accu( cols );
    boost::mpl::for_each<typename T::columns, detail::wrap<boost::mpl::placeholders::_1> >( accu );

    detail::table_constraint_creator accu2( cols );
    boost::mpl::for_each<typename T::constraints>( accu2 );

    return QLatin1Literal( "CREATE TABLE " ) % T::tableName() % QLatin1Literal( " (\n" ) % cols.join( QLatin1String( ",\n" ) ) % QLatin1Literal( "\n)" );
}
PCSSqlSelectQueryBuilder qb = PCSSqlSelectQueryBuilder();
    qb.setTable(Handover);
    qb.addAllColumns();

    PCSSqlCondition cond( PCSSqlCondition::And);
    cond.addValueCondition( Wards.short_desc, PCSSqlCondition::Is, SqlNull );
    cond.addValueCondition( Wards.description, PCSSqlCondition::LessOrEqual,
                        QString::fromLatin1("foo") );
    qb.whereCondition().addCondition( cond );
    qb.whereCondition().addColumnCondition( Wards.contact_tel, PCSSqlCondition::Greater,
                        Wards.contact_fax );
    qb.whereCondition().setLogicOperator( PCSSqlCondition::Or );

    PCSSqlSelectQueryBuilder qb2;
    qb2.setTable( QL1S("table2") );
    qb2.addColumn( QL1S("col2_2") );

    PCSSqlSelectQueryBuilder qbCombinedAll;
    qbCombinedAll.combineQueries(qb, qb2, PCSSqlSelectQueryBuilder::UnionAll);
Select in C++:

```cpp
SqlQuery query =

Sql::select(BedonWard.id).from(BedonWard).where(BedonWard.fk_tblWards_id == wardID && isNull(BedonWard.deleted));

Sql::select(Staff.id, StaffGrades.description).from(Staff).leftOuterJoin(StaffGrades, Staff.StaffGrade == StaffGrades.id);

Sql::select(Staff.id).from(Staff).orderBy(Staff.StaffSurname, Qt::DescendingOrder).queryBuilder();
```
Customer: “What if I want to have a QML UI, a native iPhone UI, a native Metro UI and an HTML5 UI front the same application?”

You, reasonably: “Let me develop all of it and we’ll sort it out, implementation detail.”

Customer: MWAH-HA-HA-HAAA!
Native UI

The Solution:

- Implement backend functionality in Qt
- Expose REST interface locally (or even SOAP)
- Pretend it's development against a web service

=> no need to buy an iPhone, leave that app to the dude with the fixie bike
The problem:
- QML application
- One central list view
- 1,000,000 items
- Smooth scrolling
- 300 Mhz
- No GPU
- No FPU
“Sometimes you have to be really high, to see how small you really are, I'm going home now.”

- Felix Baumgartner
It's just a list view ...

The solution:

- Custom QListView overlay
- Event filters that scare children
- Manually positioned at fixed pixel position
- Custom QAbstractItemModel
- No regard for layering
- Sliding window of cached, reused items
- Hints from the scrollbars
- A lot of callgrind quality time
It's just a list view
Thanks! Questions?

@tilladam
@mirkoboehm