

Qt 3.2 Whitepaper

Trolltech AS

www.trolltech.com

(株) SRA

www.sra.co.jp

2004 年 3 月 27 日

概要

このホワイトペーパーでは、C++ ツールキット Qt の解説をおこないます。Qt はマルチプラットフォームの GUI アプリケーション開発を支援するツールキットで、「write once, compile anywhere」を実現します。1 種類のソースファイルを記述しておけば、再コンパイルするだけで、Windows 95 から XP、Mac OS X、UNIX/Linux、Solaris、HP-UX やその他の Unix と X11 環境で動作するアプリケーションが作成できます。Qt アプリケーションは Qt の組み込み版である Qt/Embedded でコンパイルし、実行することもできます。Qt は「シグナルとスロット」と呼ばれる独自のオブジェクト間通信を採用しています。また、2D および 3D グラフィクス、国際化、XML といったプログラミング技法も忠実にサポートします。Qt アプリケーションは Qt Designer と呼ばれるビジュアルツールを併用して開発することもできます。

目次

1	はじめに	4
1.1	概要	4
2	ウィジェット	6
2.1	「Hello」サンプル	7
2.2	組み込みウィジェット	7
2.3	カスタムウィジェット	9
3	シグナルとスロット	12
3.1	シグナルとスロットのサンプル	13
3.2	メタオブジェクトコンパイラ (moc)	14
4	GUI アプリケーション	16
4.1	メインウィンドウクラス	16
4.1.1	メインウィンドウ	16
4.1.2	メニュー	17
4.1.3	ツールバー	18
4.1.4	バルーンヘルプ	18
4.1.5	アクション	18
4.1.6	センターウィジェット	19
4.2	マルチドキュメントインターフェイス (MDI)	19
4.3	ダイアログ	19
4.4	ドッキングウィンドウ	22
4.5	設定	23
4.6	マルチスレッド	23
5	Qt Designer	24
5.1	Qt Assistant	26
5.2	GUI アプリケーションのサンプル	27
6	2D/3D グラフィクス	29
6.1	2D グラフィクス	29
6.1.1	画像	29
6.1.2	描画	29
6.1.3	描画デバイス	31
6.1.4	キャンバス	31
6.2	3D グラフィクス	32
6.3	3D グラフィクスのサンプル	33
7	データベース	37
7.1	SQL コマンドの実行	37
7.2	データベースウィジェット	39

8	国際化	41
8.1	Unicode	41
8.2	テキスト入力とレンダリング	41
8.3	アプリケーションの翻訳	42
8.4	Qt Linguist	42
9	スタイルとテーマ	44
9.1	組み込みスタイル	44
9.2	スタイル対応ウィジェット	44
9.3	スタイルのカスタマイズ	44
10	レイアウト	46
10.1	組み込みレイアウトマネージャ	46
10.2	レイアウトのネスト	47
10.3	レイアウトのカスタマイズ	48
10.4	イベント	50
10.5	イベントの作成	50
10.6	イベントの送信	50
11	入出力とネットワーク	51
11.1	ファイル I/O	51
11.2	XML	52
11.3	プロセス間通信	52
11.4	ネットワーク	52
12	コレクションクラス	55
12.1	値ベースのコレクションクラス	55
12.2	ポインタベースのコレクションクラス	56
13	プラグインと動的ライブラリ	57
13.1	プラグイン	57
13.2	動的ライブラリ	58
14	プラットフォーム固有の拡張	59
14.1	ActiveQt	59
14.2	Motif	60
15	Qt のアーキテクチャ	61
15.1	Microsoft Windows	61
15.2	X11	62
15.3	Mac OS X	62
15.4	Embedded Linux	63
16	Qt を使った開発	64

1 はじめに

Qt はマルチプラットフォームの GUI アプリケーションを開発するための C++ ツールキットです。Qt には C++ クラスライブラリだけでなく、アプリケーションの迅速でシンプルな開発を支援するためのツールも含まれています。マルチプラットフォーム対応、国際化対応により、Qt を使ったアプリケーションは、ターゲット市場を最大限広げることができます。

Qt は 1995 年から数々の商用アプリケーションの心臓部として利用されてきました。AT&T、IBM、NASA、Xerox をはじめとして、数多くの小さな会社や組織でも広く使われてきました。Qt 3.2 では従来のバージョンに豊富な機能と新しいクラスが追加されましたが、使いやすさと性能は少しもそなわれていません。Qt のクラスは、高度な機能を持っているため開発者の負担が軽減され、首尾一貫したインターフェイスを備えているため学習も容易です。Qt はすでに、そして常に完全なオブジェクト指向を採り入れています。

このホワイトペーパーでは、Qt のツールと機能について概要を説明します。それぞれのセクションは技術的な解説以外の説明から始め、次に技術的詳細を徐々に深く掘り下げていきます。コードの断片を示し、サンプルとして、完全に動作する小さなアプリケーションを示します。Qt の 30 日間試用版は次の URL からダウンロードできます。

<http://www.trolltech.com>

1.1 概要

Qt には、標準的な GUI 機能を提供する豊富なウィジェット (Windows でいう「コントロール」) が用意されています。Qt では、オブジェクト間通信の画期的な機構である「シグナル」と「スロット」を採用しています。これは、安全性が低い従来のコールバック機構を置き換えるものです。また、マウスのクリックやキーの押下などを処理する従来型のイベントモデルも用意されています。Qt では、メニュー、コンテキストメニュー、ドラッグ&ドロップ、ドッキングツールバーなど、現代的なマルチプラットフォームの GUI アプリケーションに要求されるユーザインターフェイスを簡単に利用できます。

直観的な命名規則と一貫したプログラミング思想により、シンプルにコードを記述できます。また、グラフィカルなユーザインターフェイス作成ツール Qt Designer も用意されており、Qt の強力なレイアウト機構と絶対位置指定のいずれかによりユーザインターフェイスを作成できます。

Qt は 2D グラフィクスと 3D グラフィクスを強力にサポートします。Qt は、プラットフォームに依存しない OpenGL プログラミングツールキットとしてデファクトスタンダードの地位を獲得しています。

標準的なデータベースを使った、プラットフォームに依存しないデータベースアプリケーションを作成することもできます。Qt では、Oracle、Microsoft SQL Server、Sybase Adaptive Server、IBM DB2、PostgreSQL、MySQL、ODBC 準拠データベースのネイティブドライバをサポートしています。Qt のデータベース機能は Qt Designer に完全に統合され、データベースのデータをその場で参照することができます。Qt にはデータベース用のウィジェットがあらかじめ用意されており、組み込みウィジェットやカスタムウィジェットをデータベース対応にすることもできます。

Qt で作成したプログラムは、Qt のスタイルとテーマ機能を使うことで、サポートされているすべてのプラットフォームでネイティブのルック&フィールを持たせることができます。1 種類のソースコードを再コンパイルするだけで、Windows 95 から XP、Mac OS X、UNIX/Linux、Solaris、HP-UX やその他の UNIX と X11 環境で動作するアプリケーションを作成できます。Qt アプリケーションは Qt の組み込み版である Qt/Embedded 上でコンパイル、実行することができます。Qt のビルドツール qmake を使え

ば、目的のプラットフォームに応じて、適切な Makefile や .dsp ファイルを生成することができます。Qt のアーキテクチャは、Qt が使われているプラットフォームの機能を最大限に引き出すように設計されているため、Windows、Mac OS X、UNIX それぞれ単一のプラットフォームで動作するアプリケーションの開発にも Qt を使う顧客も数多くいます。Qt のアプローチの方が好ましいと判断されたためです。Qt は、Windows の ActiveX や UNIX の Motif など、重要なプラットフォーム固有の機能もサポートしています。

Qt は、ツールキット全体にわたって Unicode を採用しており、国際化を強力にサポートしています。翻訳作業の支援のために、Qt Linguist をはじめとするさまざまなツールが用意されています。アラビア語、中国語、英語、ヘブライ語、日本語、ロシア語他、Unicode でサポートされている言語を簡単に混在させることができます。

Qt は、さまざまな技術領域に特化したクラスも用意されています。たとえば、Qt の XML モジュールは、SAX と DOM のパーサを含みます。STL 互換のコレクションクラスを使って、オブジェクトをメモリ上に格納することができます。標準的なプロトコルを使ったローカルファイルやリモートファイルの処理は、Qt のファイル I/O クラスとネットワーククラスを使っておこないます。

Qt アプリケーションはプラグインと動的ライブラリを使って機能を拡張することができます。プラグイン機構を使えば、Codec、データベースドライバ、画像フォーマット、スタイル、ウィジェットを追加することができます。動的ライブラリを使えば、追加できる機能に制限はありません。プラグインと動的ライブラリは、どちらも独立した製品として販売することもできます。

Qt は完成された C++ ツールキットで、世界中で広く使われています。商用製品にも多く使われていますが、Qt の無償版は、UNIX/Linux のデスクトップ環境である KDE のベースとしても採用されています。Qt のマルチプラットフォーム対応のビルドシステム、フォームのビジュアルなデザイン、洗練された API により、アプリケーション開発が楽しくなるのです。

オンラインリファレンス

<http://www.trolltech.com/references/customers/>

<http://www.trolltech.com/references/partners/>

2 ウィジェット

Qt には豊富な種類のウィジェット (ボタン、スクロールバーなど) が用意されており、ほとんどのケースではこれで十分です。Qt のウィジェットは柔軟性が高く、特殊な用途向けにサブクラス化することも容易です。

Qt には、通常必要なウィジェットがすべて用意されています。ウィジェットとは、ユーザインターフェイスを構成する GUI 要素のことです。たとえば、ボタン、メニュー、スクロールバー、メッセージボックス、それにアプリケーションのメインウィンドウなどはすべてウィジェットです。Qt のウィジェットは、恣意的に「コントロール」と「コンテナ」とに分類することはしません。すべてのウィジェットはコントロールにもコンテナにもなることができます。既存の Qt ウィジェットをサブクラス化することで、カスタムウィジェットを作成することも容易です。また、ごく特殊なケースとして、必要があればスクラッチからカスタムウィジェットを作成することもできます。

ウィジェットは、QWidget が、QWidget のサブクラスのインスタンスで、カスタムウィジェットはウィジェットをサブクラス化することで作成します。

ウィジェットには任意の数の子ウィジェットを持たせることができます。子ウィジェットは親ウィジェットの領域内に表示されます。親ウィジェットを持たないウィジェットを「トップレベルウィジェット」と呼びます (これがアプリケーションウィンドウとなります)、通常、デスクトップ環境のタスクバーに表示されますが、トップレベルウィジェットに関しては何の制限もありません。どのウィジェットでもトップレベルウィジェットになることができ、任意のウィジェットは任意のウィジェットの子となることができます。親ウィジェットの領域内で子ウィジェットが表示される位置は、レイアウトマネージャによって自動的に設定されますが、自分で指定することもできます。親ウィジェットが入力禁止になったり、隠されたり、破棄されると、すべての子ウィジェットについて同じ処理がおこなわれます。

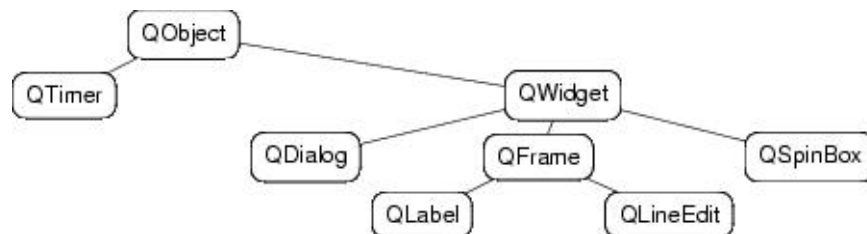


図 1. QWidget クラス階層の一部

ラベル、メッセージボックス、ツールチップなどは、色やフォント、言語を複数混在させることができます。Qt のテキストレンダリングウィジェットは、HTML のサブセット機能を使って複数言語をリッチテキストで表示することができます。詳細は、「8.2. テキスト入力とレンダリング」を参照してください。

2.1 「Hello」サンプル



図 2. Hello, Qt!

「Hello, Qt!」と表示するプログラムのソースコード全体を以下に示します。

```
#include <qapplication.h>
#include <qlabel.h>

int main( int argc, char **argv )
{
    QApplication app( argc, argv );

    QLabel hello( "<font color=blue>Hello <i>Qt!</i>"
                 "</font>", 0 );
    app.setMainWidget( &hello );
    hello.show();

    return app.exec();
}
```

2.2 組み込みウィジェット

Qt の主なウィジェットを以下に示します。Windows スタイルで表示したものです。

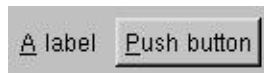


図 3. QLabel と QPushButton ウィジェットを QHBoxLayout でレイアウト

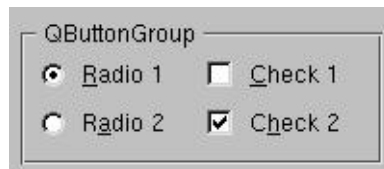


図 4. 2 つの QRadioButton と 2 つの QCheckBox を QButtonGroup でレイアウト

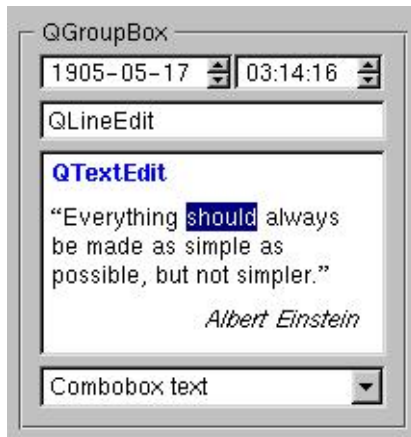


図 5. QDateTimeEdit、QLineEdit、QTextEdit、QComboBox を QGroupBox でレイアウト

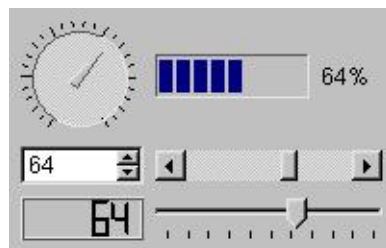


図 6. QDial、QProgressBar、QSpinBox、QScrollBar、QLCDNumber、QSlider を QGrid でレイアウト

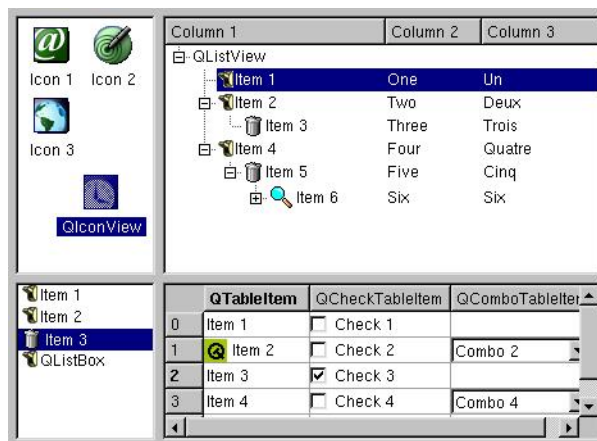


図 7. QIconView、QListView、QListBox、QTable を QGrid でレイアウト

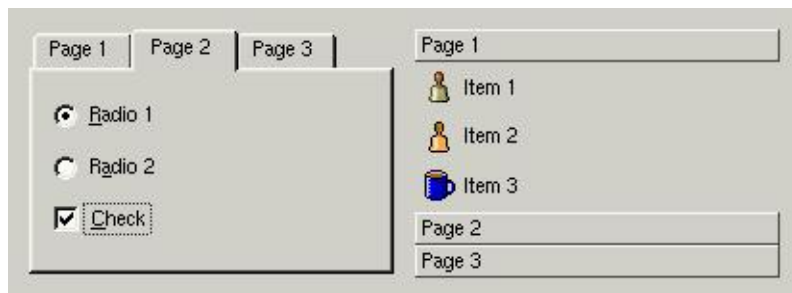


図 8. QTabWidget、QToolBox を QHBox でレイアウト

QComboBox、QLineEdit、QSpinBox は、QValidator のサブクラスを使って入力制限と入力チェックをおこなうことができます。入力チェックには正規表現を使うこともできます。

大量のデータを表示するウィジェット (QTable、QListView、QTextEdit など) は、QScrollView を継承しており、スクロールバーが自動的に表示されます。

QMainWindow、QMenuBar、QToolBar の使い方は「4. GUI アプリケーション」で説明します。QMessageBox、QFileDialog、QWizard やその他のダイアログボックスは「4.3. ダイアログ」で説明します。QSplitter は「10. レイアウト」、QCanvas、QGLWidget は「6. 2D/3D グラフィクス」でそれぞれ説明します。

QRadioButton と QCheckBox のスクリーンショット (図 4) は、次のようにして作成します。

```
parent = new QButtonGroup( 2, Qt::Vertical, "QButtonGroup" );
radio1 = new QRadioButton( "&Radio 1", parent );
radio2 = new QRadioButton( "R&adio 2", parent );
radio1->setChecked( true );
check1 = new QCheckBox( "&Check 1", parent );
check2 = new QCheckBox( "C&heck 2", parent );
check2->setChecked( true );
```

2.3 カスタムウィジェット

QWidget か、QWidget のサブクラスをサブクラス化することで、カスタムウィジェットを作成することができます。デジタル時計ウィジェットの完全なコードを示し、カスタムウィジェットの作成方法を説明します。



図 9. デジタル時計ウィジェット

Clock ウィジェットは現在の時刻を LCD で表示し、時刻を自動的に更新するというものです。時と分を区切るコロンが 1 秒ごとに点滅します。

Clock ウィジェットは、clock.h で次のように宣言されます。

```

#include <qlcdnumber.h>
class Clock : public QLCDNumber
{
public:
    Clock( QWidget* parent = 0, const char* name = 0 );
protected:
    void timerEvent( QTimerEvent* event );
private:
    void showTime();
    bool showingColon;
};

```

Clock クラスは QLCDNumber ウィジェットから LCD 表示機能を継承します。コンストラクタは、ウィジェットクラスの典型的なコンストラクタです。parent パラメータと name パラメータはオプションです (name パラメータはテストやデバッグを容易にするために設定するものです)。timerEvent() メソッドは QObject からの継承メソッドで、一定時間ごとにシステムから呼び出されます。

clock.h で宣言された関数は clock.cpp で定義されます。

```

#include <qdatetime.h>
#include "clock.h"

Clock::Clock( QWidget* parent, const char* name )
    : QLCDNumber( parent, name ), showingColon( true )
{
    showTime();
    startTimer( 1000 );
}

void Clock::timerEvent( QTimerEvent* )
{
    showTime();
}

void Clock::showTime()
{
    QString time = QTime::currentTime().toString().left( 5 );
    if ( !showingColon )
        time[2] = ' ';
    display( time );
    showingColon = !showingColon;
}

```

コンストラクタでは、showTime() を呼び出して時計を現在時刻に初期化し、timerEvent() を 1000 ミリ秒ごとに呼び出して LCD 表示を更新するように指定しています。

showTime() メソッド内では、現在時刻を与えて QLCDNumber::display() を呼び出しています。showTime() を呼び出すごとにコロンの空白を交互に表示させることで、コロンを点滅させます。

clock.h と clock.cpp は、カスタムウィジェット Clock の完全な宣言と定義です。こうして作成したウィジェットは、次のようにすぐ使えるようになります。

```

#include <qapplication.h>
#include "clock.h"
int main( int argc, char **argv )
{
    QApplication app( argc, argv );
    Clock *clock = new Clock;
}

```

```
    app.setMainWidget( clock );
    clock->show();
    return app.exec();
}
```

このサンプルでは、ウィジェットを1つ (Clock ウィジェット) だけを使い、子ウィジェットは使いませんでした。複雑なカスタムウィジェットの場合、ウィジェットとレイアウトを組み合わせることで作成することになるでしょう。

スクラッチからカスタムウィジェットを作成することもできます。たとえばアナログ時計ウィジェットを作成したい場合、時計面と時計の針を描くコードは、基本クラスで実装されている機能ではなく、自分で実装する必要が生じるでしょう。こういった処理については、「6.1. 2D グラフィクス」で説明します。

オンラインリファレンス

<http://doc.trolltech.com/3.2/qwidget.html>

3 シグナルとスロット

シグナルとスロットはオブジェクト間通信の機構です。理解しやすく使いやすく、Qt Designer で完全にサポートされています。

GUI アプリケーションは、ユーザの動作に応答するものです。たとえばユーザがメニュー項目やツールバーのボタンをクリックすると、アプリケーションは何らかの処理を実行します。もう少し一般的に言うと、あらゆる種類のオブジェクト同士を互いに通信させたいということになります。そのために、何らかのイベントと何らかのコードを関連づけなければなりません。従来のツールキットの機構は、型保障がおこなわれず（つまりクラッシュしやすいということ）、柔軟性に欠け、オブジェクト指向ではありませんでした。 Trolltech 社は、「シグナルとスロット」と呼ばれる機構を採用しました。シグナルとスロットは強力なオブジェクト通信間の機構で、これまでのツールキットで使われてきた、古くさいコールバックやメッセージマップを完全に置き換えるものです。シグナルとスロットは柔軟性があり、完全にオブジェクト指向で、C++ で実装されます。



図 10. シグナルとスロットの接続の概要

従来のコールバック機構で、ボタンに何らかの処理を結びつける場合、関数へのポインタをボタンに渡しておく必要があります。ボタンがクリックされると関数が呼び出されます。しかし、この方法では関数が呼び出されるときに、パラメータに正しい型が使われる保証ができないため、アプリケーションがクラッ

シュシやすくなります。コールバック機構のもう 1 つの問題点は、特定の機能が GUI 要素に密接に結びつけられてしまうという点です。このため、クラスを独立して開発することが難しくなります。

Qt のシグナルとスロット機構は違います。Qt ウィジェットは、何らかのイベントが発生したときシグナルを送出します。たとえば、ボタンがクリックされると "clicked" シグナルが送出されます。プログラマは、シグナルを処理する関数 (スロットと呼びます) を作成し、connect() メソッドを使ってシグナルとスロットを結びつけます。シグナルとスロット機構では、シグナルとスロットは互いについて何も知る必要がありません。そのため、高度に再利用可能なクラスを容易に開発することができます。シグナルとスロットは型保障がなされるので、誤った型が渡されても、アプリケーションをクラッシュさせるのではなく、警告が報告されるだけです。

たとえば、「Quit」ボタンの clicked() シグナルをアプリケーションの quit() スロットに接続したとしましょう。ユーザが「Quit」ボタンをクリックするとアプリケーションが終了します。コードで書くと、次のようになります。

```
connect( button, SIGNAL(clicked()), QApplication, SLOT(quit()) );
```

アプリケーションの実行中、いつでも接続したり接続を解除することができます。

シグナルとスロットは C++ 構文の自然な拡張で記述します。C++ のオブジェクト指向の能力を最大限引き出すことができます。シグナルとスロットは型保障がおこなわれ、オーバーロードしたり再定義することもでき、public、protected、private いずれでも宣言できます。

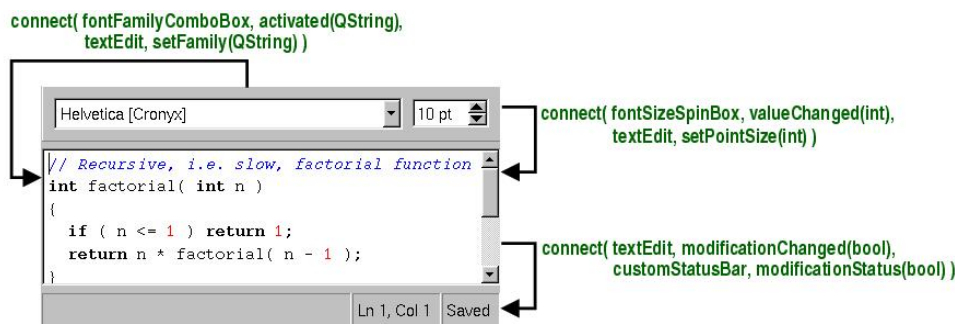


図 11. シグナルとスロットの接続の例

3.1 シグナルとスロットのサンプル

シグナルとスロットを利用するには、QObject または QObject のサブクラスを継承する必要があります。また、クラス宣言中に Q_OBJECT マクロを記述します。シグナルはクラス宣言の signals セクションに記述し、slot は public slots、protected slots、private slots のいずれかのセクションに記述します。

QObject のサブクラスの例を示します。

```
class BankAccount : public QObject
{
    Q_OBJECT

public:
    BankAccount() { curBalance = 0; }
    int balance() const { return curBalance; }
public slots:
    void setBalance( int newBalance );
```

```

signals:
    void balanceChanged( int newBalance );
private:
    int curBalance;
};

```

BankAccount クラスは一般的な C++ クラスの同様に、コンストラクタ、取得関数 (balance())、設定関数 (setBalance()) を持っています。

BankAccount クラスには balanceChanged() シグナルが宣言されています。このシグナルは、口座の残高が変更されたことを通知するものです。シグナルが送出されると、接続されているスロットが実行されます。

設定関数は public slots セクションで宣言されていますから、スロットです。しかし、スロットは通常のメンバ関数となんら変わることがなく、他の関数から呼び出すことも、シグナルに接続することもできます。

setBalance() スロットの定義は次の通りです。

```

void BankAccount::setBalance( int newBalance )
{
    if ( newBalance != curBalance ) {
        curBalance = newBalance;
        emit balanceChanged( curBalance );
    }
}

```

次の行

```

emit balanceChanged( curBalance );

```

がシグナルの送出です。シグナルの引数は変更後の残高です。emit キーワードは、signals と slots と同様 Qt 特有のキーワードで、C++ プロセッサを通して通常の C++ 構文に変換されます。

2 つの BankAccount オブジェクトを接続する例です。

```

BankAccount x, y;
connect( &x, SIGNAL(balanceChanged(int)), &y, SLOT(setBalance(int)) );
x.setBalance( 2450 );

```

インスタンス x の残高を 2450 に変更すると、balanceChanged() シグナルが送出され、インスタンス y の setBalance() スロットで受け取られます。その結果、インスタンス y の残高が 2450 に変更されます。

1 つのシグナルは複数のスロットに接続することができ、また逆に 1 つのスロットを複数のシグナルに接続することができます。接続するシグナルとスロットのパラメータ型は一致していなければなりません。しかし、シグナルの引数より少ない引数のスロットを接続することができます。その場合、余分の引数は無視されるだけです。

3.2 メタオブジェクトコンパイラ (moc)

シグナルとスロットは標準的な C++ の範囲で実装されています。ただし、C++ プリプロセッサの Meta Object Compiler (moc) を使って変換する必要があります。moc は Qt ツールキットに同梱されています。

moc はアプリケーションのヘッダファイルを読み出し、シグナルとスロットを実現するためのソースコードを生成します。moc は makefile ジェネレータである qmake を使って自動的に起動されるため、プログラマは生成されたソースコードを編集することはおろか、見る必要すらありません。

moc はシグナルとスロット用のソースコードを生成するだけでなく、Qt の翻訳機構、プロパティ、拡張された実行時型情報の処理もおこないます。また、moc を使うことで C++ プログラムをマルチプラットフォーム対応にすることができます。

オンラインリファレンス

<http://doc.trolltech.com/3.2/object.html>

<http://doc.trolltech.com/3.2/signalsandslots.html>

<http://doc.trolltech.com/3.2/moc.html>

4 GUI アプリケーション

Qt を使えば、現代的な GUI アプリケーションを早く、簡単に作成することができます。GUI アプリケーションは手作業で開発することもできますが、Qt のビジュアルデザインツールである Qt Designer を使って作成することもできます。

Qt には、現代的な GUI アプリケーションを作成するために必要なクラスや関数がすべてそろっています。メインウィンドウを持ち、クライアント領域の周囲にメニューバー、ツールバー、ステータスバーを持つタイプのアプリケーション、ボタンやタブを持ち、オプションや情報を表示するダイアログスタイルのアプリケーションのどちらでも作成できます。また、Qt は SDI (Single Document Interface) と MDI (Multi Document Interface) の両方をサポートします。ドラッグ&ドロップとクリップボードもサポートします。

ツールバーはツールバー領域内部で移動することも、ウィンドウの他の辺に移動することも、フローティングツールバーとすることもできます。こうした機能は完全に Qt 内部で実装しており、プログラマは 1 行もコードを書く必要はありませんし、必要に応じてツールバーの動作を制限することもできます。

Qt はプログラミングをシンプルにします。たとえば、メニュー項目とツールバーボタンとキーボードアクセラレータが同じ動作を実行する場合、コードは 1 箇所だけに記述すればよいのです。

メッセージボックスと通常使われる標準的なダイアログボックスはすべて備えているため、ユーザに何らかの問い合わせをおこなう処理を簡単に記述できます。たとえばファイルやフォルダを選択する、フォントを選択する、色を選択するなどです。実際、メッセージボックスや標準ダイアログを表示するには、1 つの静的関数を呼び出すだけです。

Qt では、ユーザの設定や最近使ったファイルリスト、ウィンドウのサイズ、ツールバーの位置とサイズといったアプリケーションデータをプラットフォームに依存しない形で保存することができます。

4.1 メインウィンドウクラス

4.1.1 メインウィンドウ

QMainWindow クラスは一般的なアプリケーションのメインウィンドウを作成するためのクラスです。メインウィンドウは標準ウィジェットから構成されます。メインウィンドウの最上部にはメニューバーが配置されます。メニューバーの下にツールバーが置かれます。ツールバーはツールバー領域のどの位置にでも移動でき、メインウィンドウの上下左右のどの辺にでも移動することができます。また、ツールバー領域からドラッグすることで、独立したフローティングツールバーとすることもできます。メインウィンドウの最下部にはステータスバーが表示されます。ステータスバーの上にボトムツールバーが表示されることもあります。ウィンドウのクライアント領域には、SDI アプリケーションの場合任意のウィジェットを、MDI アプリケーションの場合 QWorkspace を表示します。ツールチップと「これは何」ヘルプは、どちらも GUI 要素に関するバルーンヘルプを表示するものです。

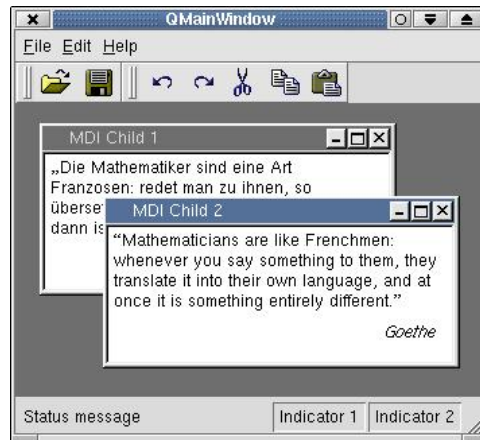


図 12. アプリケーションのメインウィンドウ

4.1.2 メニュー

QPopupMenu ウィジェットはメニュー項目を縦に並べたリストを表示するウィジェットです。ポップアップメニューはメニューバーに表示することも、他のポップアップメニューのサブメニューとして表示することも、単独で表示する (いわゆるコンテキストメニュー) こともできます。ポップメニューには切り離しハンドルを持たせることもできます。

メニュー項目にはアイコン、チェックボックス、アクセラレータを表示することもできます。メニュー項目は、通常なんらかの動作 (たとえば「保存」など) を実行します。セパレータ項目は関連する動作を視覚的にグループ化するための水平線です。

「新規作成」、「開く」、「終了」を持つ「ファイル」メニューを作成するサンプルです。

```
QPopupMenu *fileMenu = new QPopupMenu( this );
fileMenu->insertItem( "&New", this, SLOT(newFile()), CTRL+Key_N );
fileMenu->insertItem( "&Open...", this, SLOT(open()), CTRL+Key_O );
fileMenu->insertSeparator();
fileMenu->insertItem( "E&xit", qApp, SLOT(quit()), CTRL+Key_Q );
```

メニュー項目が選択されると、対応するスロットが実行されます。

QMenuBar クラスはメニューバーの実装です。メニューバーは自動的に親ウィジェット (通常は QMainWindow) の最上部に配置され、メニューバーの項目数が親ウィンドウの幅より大きい場合、自動的に複数行に分割されます。Qt の組み込みレイアウトマネージャは、メニューバー領域を考慮してウィジェットのレイアウトをおこないます。Mac OS X では、メニューはスクリーンの最上部に表示されます。

「ファイル」、「編集」、「ヘルプ」メニューを持つメニューバーを作成するサンプルです。

```
QMenuBar *bar = new QMenuBar( this );
bar->insertItem( "&File", fileMenu );
bar->insertItem( "&Edit", editMenu );
bar->insertItem( "&Help", helpMenu );
```

Qt のメニュー管理は非常に柔軟です。メニュー項目の有効・無効、追加・削除は実行時に動的におこなえます。独自の表示をおこなうメニュー項目を作成するには、QCustomMenuItem クラスをサブクラス化します。

4.1.3 ツールバー

ツールバーボタンを実装するのは `QToolButton` クラスです。ツールバーボタンはアイコン、3D フレーム、ラベルを持つことができます。ツールバーボタンのトグル機能は有効・無効にすることができます。トグルボタン以外のツールバーボタンは何らかの処理を実行します。実行中、有効、無効、オン、オフの各状態ごとに異なるアイコンを表示させることもできます。アイコンを1つしか指定しなかった場合、たとえば無効のボタンをグレイアウトするなど、自動的に Qt が各状態に適した表示をおこないます。ツールバーボタンでポップアップメニューを表示することもできます。

`QToolButton` は、通常 `QToolBar` 内に順番に表示されます。ツールバーボタンはいくつでも表示することができます。ツールバーはウィンドウのどの辺にでも移動することができます。ツールバーにはほぼあらゆる種類のウィジェットを表示することができます。たとえば `QComboBox` や `QSpinBox` などです。

4.1.4 バルーンヘルプ

現代的なアプリケーションでは、ユーザインターフェイス要素の目的を簡単に説明するバルーンヘルプを持っています。Qt では、ツールチップと「これは何？」という2種類のバルーンヘルプを用意しています。

ツールチップは、小さく、通常は淡い黄色の長方形で、ウィジェットの上にマウスポインタを移動すると自動的に表示されるヘルプです。ツールチップは、よくツールバーボタンの説明に使われます。ツールバーボタンにラベルが表示されることはほとんどないためです。「保存」ツールバーボタンにツールチップを表示するサンプルを示します。

```
QToolTip::add( saveButton, "Save" );
```

ツールチップが表示されたときにステータスバーにも同じ説明が表示されるように、長い文章を指定することも可能です。

「これは何？」ヘルプはツールチップと似ていますが、自動的に表示されるのではなく、ユーザが指示した場合にだけ表示されるという点が異なります。たとえば `Shift+F1` キーを押してから目的のウィジェットやメニュー項目をクリックする場合などです。「これは何？」ヘルプは、ツールチップの説明よりも長いことが一般的です。「保存」ツールバーボタンに「これは何？」ヘルプを表示するサンプルです。

```
QWhatsThis::add( saveButton, "Saves the current file." );
```

`QToolTip` と `QWhatsThis` クラスは、ウィジェット内をクリックされた位置によって異なるテキストを表示するなど、標準と異なる動作を記述できるように、仮想関数を持っています。

4.1.5 アクション

一般的なアプリケーションでは、ある動作を実行するためにさまざまな手段を用意しています。たとえば、「保存」という動作はメニュー項目（「ファイル—保存」メニュー）とツールバー（フロッピーディスクのアイコンのボタン）、それにキーボードアクセラレータ (`Ctrl+S`) から実行することができます。`QAction` クラスは、この方法をカプセル化したものです。`QAction` クラスを使って、動作を1箇所だけで定義します。以下に示すサンプルコードは「保存」処理をメニュー、ツールバーボタン、アクセラレータキーに割り当てるものです。メニュー項目にもツールバーボタンにもバルーンヘルプを指定しています。

```
QAction *saveAct = new QAction( "Save", saveIcon, "&Save",
```

```
CTRL+Key_S, this );
connect( saveAct, SIGNAL(activated()), this, SLOT(save()) );
saveAct->setWhatsThis( "Saves the current file." );
saveAct->addTo( fileMenu );
saveAct->addTo( toolbar );
```

QAction には、コードの重複を避けるという意味のほかに、メニュー項目の状態とツールバーボタンの状態を同期させること、必要なときにツールチップを表示するという機能もあります。アクションを無効にすると、対応するメニュー項目とツールバーボタンも無効になります。また、トグルツールバーボタンをクリックすると、対応するメニュー項目もそれに同期してチェック・非チェック状態になります。

4.1.6 センターウィジェット

QMainWindow のクライアント領域にはどのウィジェットでも指定することができます。たとえばテキストエディタの場合、QTextEdit をセンターウィジェットに指定することができます。

```
QTextEdit *editor = new QTextEdit( mainWindow );
mainWindow->setCentralWidget( editor );
```

4.2 マルチドキュメントインターフェイス (MDI)

マルチドキュメントインターフェイス (MDI) は QWorkspace クラスで実装されます。QWorkspace クラスは QMainWindow のセンターウィジェットとして指定されることが一般的です。

QWorkspace の子ウィジェットにはどのウィジェットでも指定することができます。QWorkspace の子ウィジェットは、トップレベルウィジェットと同様のフレームが描かれます。また、トップレベルウィジェットと同様、MDI の子ウィジェットでも show()、hide()、showMaximized()、setCaption() などの関数を使うことができます。

QWorkspace はウィンドウを重ねて表示する、並べて表示するなどのウィンドウ配置をおこなえます。子ウィジェットが MDI 領域からはみ出す場合にはスクロールバーが自動的に表示されます。子ウィジェットを最大化するとメニューバーにフレームボタン (最小化ボタンなど) が表示されます。

4.3 ダイアログ

GUI アプリケーションでは、何らかの処理をおこなうためにダイアログボックスを備えているものがほとんどです。Qt にはよく使われるダイアログボックスが用意されており、ダイアログを簡便に使うための関数も用意されています。Qt の標準的なダイアログボックスの例を下の図に示します。これ以外にも色の選択ダイアログや印刷ダイアログといった標準ダイアログボックスが用意されています。

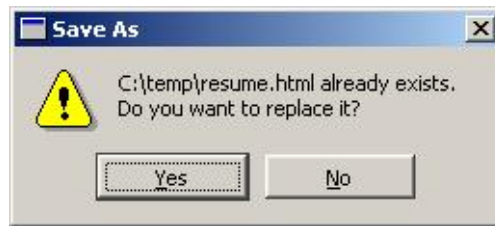


図 13. QMessageBox

QMessageBox はユーザに情報を示して簡単な選択 (「はい」 または 「いいえ」 など) をおこなわせるためのものです。

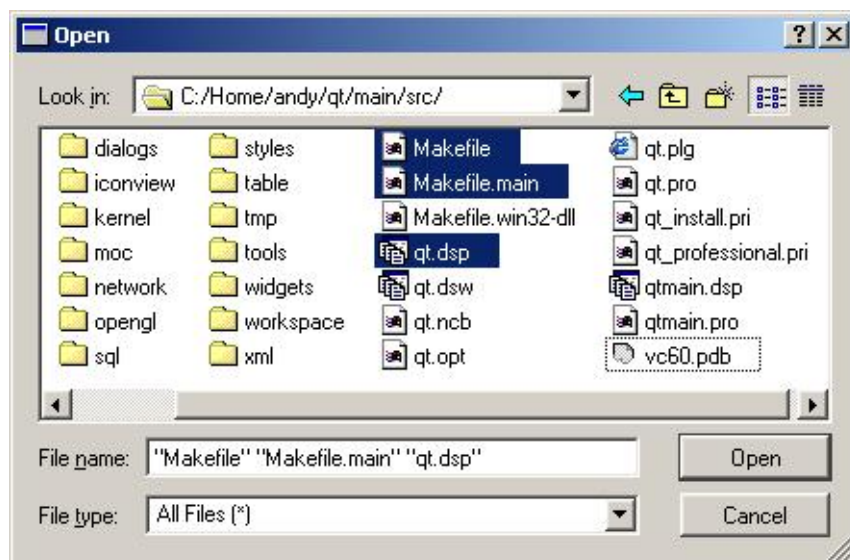


図 14. QFileDialog

QFileDialog は洗練されたファイル選択ダイアログです。1つのファイルを選択することも複数のファイルを選択することもでき、ファイルはローカルでもリモート (FTP など) でも可能です。また、ファイルの名前変更やディレクトリの新規作成などの機能も持っています。Qt のダイアログ全般に当てはまりますが、QFileDialog はサイズ変更が可能です。このため、長いファイル名や多数のディレクトリを一度に表示することが簡単におこなえます。また、自動的に Windows や Macintosh の標準ファイルダイアログを使うように指定することもできます。

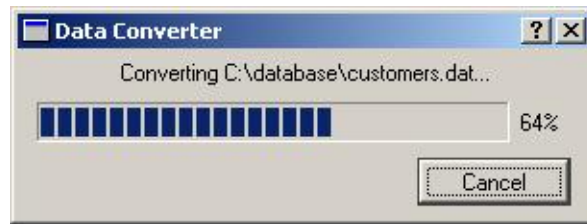


図 15. QProgressDialog

QProgressDialog はプログレスバーと「キャンセル」ボタンを表示します。



図 16. QWizard

QWizard はウィザードダイアログのテンプレートです。

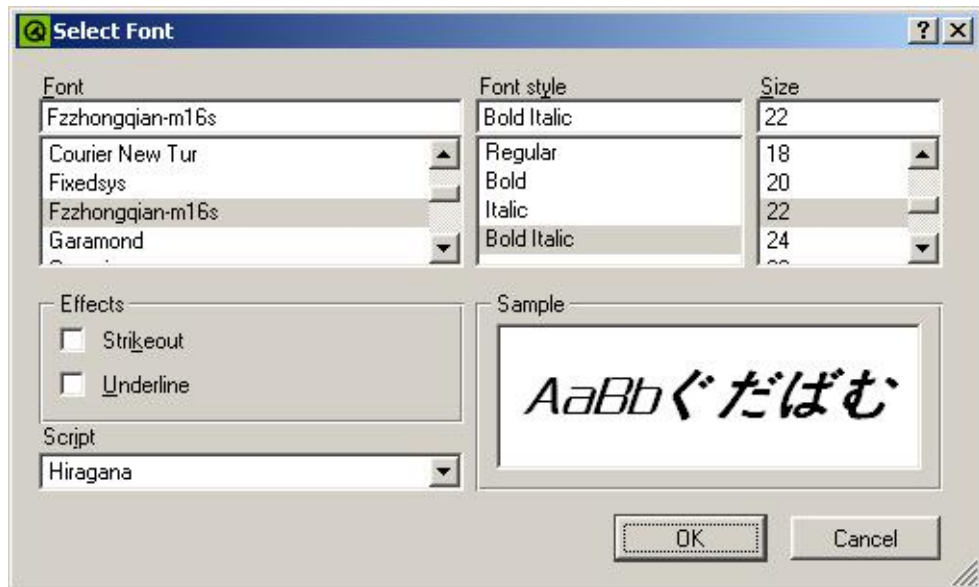


図 17. QFontDialog

QFontDialog はフォント選択に使用します。
 ダイアログは次の 3 種類のモードを持っています。

1. モーダルダイアログ。同じアプリケーションの他のウィンドウの入力がおこなえなくなります。他のウィンドウにアクセスするには、ダイアログボックスを閉じなければなりません。
2. モードレスダイアログ。他のウィンドウの入力がおこなえます。
3. セミモーダルダイアログ。ダイアログを表示すると、すぐに呼び出し元に制御が戻ります。ユーザの視点からはモーダルダイアログと同様ですが、アプリケーションはその間処理を継続することができます。プログレスダイアログなどの場合に便利です。

モーダルダイアログの一般的な作成方法を次に示します。

```
OptionsDialog dialog( &optionsData );
if ( dialog.exec() ) {
    do_something( optionsData );
}
```

QDialog クラスをサブクラス化することで、独自のダイアログを作成することができます。QDialog は QWidget を継承しています。

4.4 ドッキングウィンドウ

ドッキングウィンドウは、ツールバー領域に格納したり、あるツールバー領域から他のツールバー領域に移動したりすることのできるウィンドウです。ドッキングウィンドウはツールバー領域から切り離すことができ、アプリケーションウィンドウの前面に表示したり、最小化することができます。ドッキングウィンドウは QDockWindow クラス、ツールバー領域は QDockArea クラスで実装されます。

Qt では、QDockWindow のサブクラスとして QToolBar を定義しています。QMainWindow は、ウィンドウの 4 つの辺にあらかじめツールバー領域を保持しています。

QDockWindow をインスタンス化し、ウィジェットを追加することでドッキングウィンドウを作成することができます。ツールバーが水平方向 (メインウィンドウの上部など) の場合ウィジェットは左から右に配置され、垂直方向 (メインウィンドウの左側など) の場合上から下に配置されます。

ドッキング領域は QMainWindow しか持たせることができないわけではありません。QDockArea ほどのウィジェットにも持たせることができ、ツールバーやドッキングウィンドウはどのツールバー領域にでも持たせることができます。

Qt Designer や Qt Linguist など、ドッキングウィンドウを拡張するアプリケーションもあります。QDockArea にはドッキングウィンドウの位置を記憶し、あとで再現する機能が備わっており、ユーザが指定したドッキングウィンドウ位置を再現するアプリケーションも簡単に作成することができます。

4.5 設定

ユーザ設定やアプリケーション設定は QSettings クラスを使ってディスク上に保存できます。Windows では、QSettings はシステムレジストリを利用します。Windows 以外のプラットフォームでは、設定はテキストファイルに保存されます。

各設定はキーを使って保存されます。たとえば /SoftwareInc/Zoomer/RecentFiles というキーは最近使ったファイルの一覧を保存します。保存できる設定はブール値、数値、Unicode 文字列、Unicode 文字列リストです。

4.6 マルチスレッド

GUI アプリケーションでは、マルチスレッドを利用する場面が多くあります。あるスレッドがユーザインターフェイスの応答を担当し、大きなファイルを読み出したり複雑な計算など、時間のかかる処理を他の複数のスレッドで実行するなどです。Qt ではマルチスレッドをサポートするように設定することができます。その場合、QThread、QThreadStorage、QMutex、QMutexLocker、QSemaphore、QWaitCondition の 6 つのクラスが利用できます。

オンラインリファレンス

<http://doc.trolltech.com/3.2/threads.html>

5 Qt Designer

Qt Designer は Qt を使って作成されたビジュアル UI エディタ、コードエディタです。いわゆる統合開発環境 (IDE, Integrated Development Environment) ではなく、Qt による開発を補助する GUI アプリケーションです。一からソースコードを書いてアプリケーションを作成することもできますが、Qt Designer を併用すると開発速度の向上が図れます。

Qt Designer を使ってフォームの設計をすることは非常に簡単です。各ウィジェットをあらわすツールボックスのボタンをクリックし、次にフォーム上でウィジェットを配置したい位置でクリックすればよいだけです。ウィジェットのプロパティはプロパティエディタで変更することができます。ウィジェットの正確な位置やサイズは気にする必要はありません。ウィジェットを選択してレイアウトを指定するだけです。たとえば、ボタンウィジェットを複数選択し、「layout horizontally」オプションを選択すれば、ボタンは水平方向にレイアウトされます。こういった方法により、フォームデザインが非常にすばやくおこなえ、ユーザがウィンドウサイズを変更してもウィジェットが適切に配置されます。Qt のレイアウト機能について詳細は「10. レイアウト」を参照ください。

Qt Designer を使えば、ユーザインターフェイスのデザインで、「コンパイル・リンク・実行」という時間のかかる方法を取らずに済みます。デザインの修正や変更がきわめて簡単になるのです。Qt Designer はプレビュー機能も持っており、フォームを他のスタイルで表示させることも可能です。たとえば Macintosh 上で開発している場合、Windows スタイルでフォームを表示することも簡単です。Qt のデータベースクラスを使って、データベース上のデータをその場でプレビューしたり編集することもできます。Qt のデータベースサポートについては「7. データベース」を参照ください。

ダイアログスタイルのアプリケーションも、メニュー、ツールバー、バルーンヘルプなどを持つ通常のスタイルのアプリケーションを作成することもできます。フォームのテンプレートもあらかじめいくつか用意されていますが、アプリケーションやアプリケーションファミリー全体を通じて一貫したデザインを持たせるために、独自のテンプレートを作成することもできます。カスタムウィジェットを作成し、Qt Designer に統合することも簡単です。

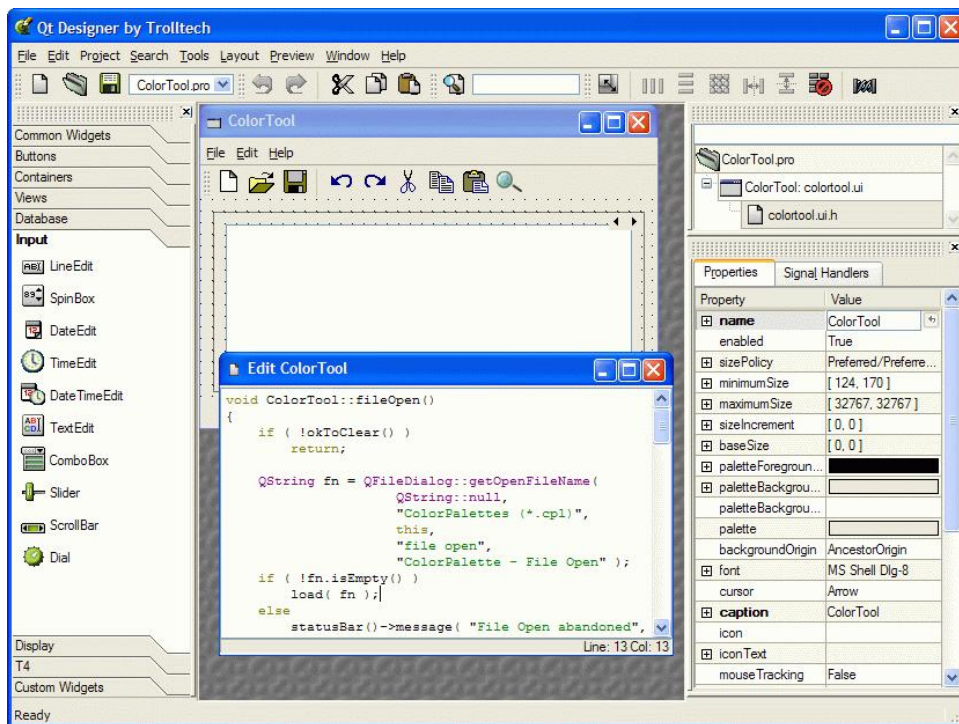


図 18. Qt Designer

Qt Designer ではアプリケーション開発はプロジェクトベースでおこないます。プロジェクトは *.pro ファイルに格納され、qmake がプロジェクトファイルを使ってメイクファイルを生成します。まずプロジェクトを新規作成し、次に必要に応じてフォームとソースファイルをプロジェクトに追加します。フォームクラスをサブクラス化することでユーザインターフェイスと機能を完全に分離することもでき、Qt Designer に内蔵されているソースコードエディタでフォームのソースを直接編集し、フォームとソースを同期させることもできます。

アプリケーションで使われるアイコンその他の画像は、プロジェクトに含まれるすべてのフォームで共有されるため、実行ファイルのサイズが小さく、またロードの時間が短縮されます。

フォームのデザインは *.ui という拡張子のファイルに XML フォーマットで格納され、uic (User Interface Compiler) を使って C++ ヘッダファイルと実装ファイルに変換されます。ビルドツール qmake が生成するメイクファイルには uic を使ったビルドルールが含まれるので、開発者が毎回直接 uic を実行する必要はありません。

通常、フォームは実行可能ファイルにコンパイルされます。しかし、アプリケーションのソースコードを利用できないエンドユーザがフォームの外見を変更したいケースもあります。このような目的のために、Qt では「ダイナミックダイアログ」をサポートしています。*.ui ファイルは実行時に動的にロードされ、完全に機能するフォームに変換されます。エンドユーザは Qt Designer を使ってアプリケーションフォームの外見を変更できるのです。ダイナミックダイアログのロードは次のように簡単です。

```
QDialog *creditForm = (QDialog *)QWidgetFactory::create( "creditform.ui" );
```

5.1 Qt Assistant

Qt Designer のオンラインヘルプは Qt Assistant を使って作成されています。Qt Assistant は Qt のすべてのドキュメントを Web ブラウザと同じフォームで表示・修正することができます。Web ブラウザとの違いは、Qt Assistant は洗練されたインデックスアルゴリズムを備えており、表示中の全ドキュメントの全文検索機能を持っていることです。

Qt のリファレンスマニュアルは全部で 1,600 の HTML ページ (印刷すると 2,500 ページにも及びます) からなり、Qt のクラスとツール、さまざまな Qt プログラミング技法の概要やチュートリアルが含まれます。

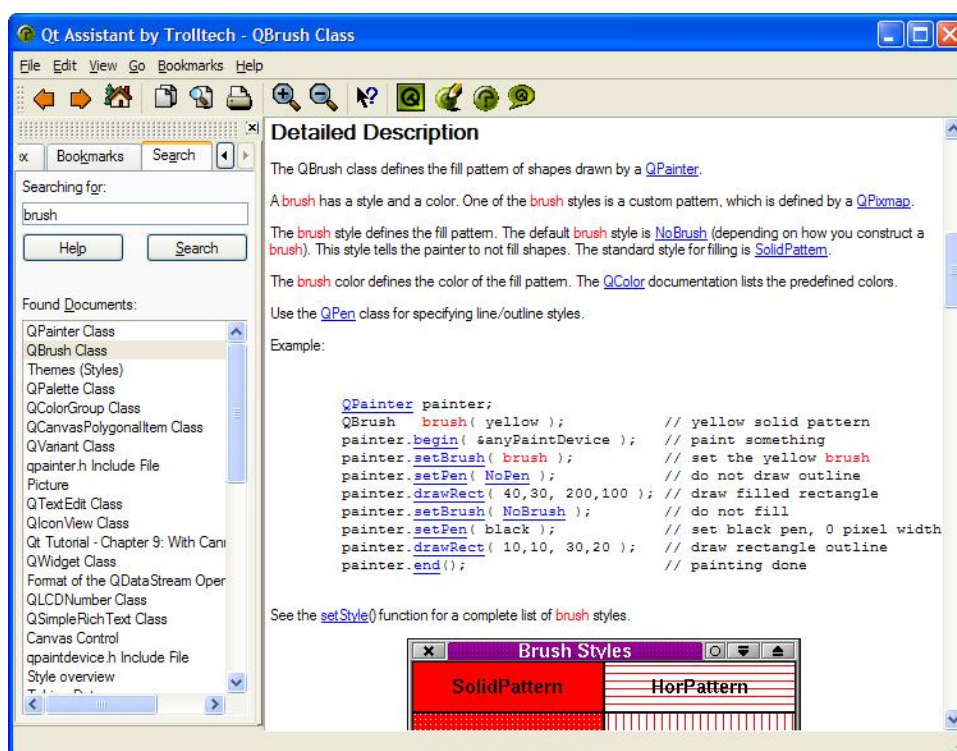


図 19. Qt Assistant

Qt Assistant はアプリケーションのドキュメントを表示するヘルプブラウザとして使うことができます。Qt Assistant は QAssistantClient クラスを使ってアプリケーションと統合することができます。Qt Assistant は QTextEdit を使って HTML ドキュメントのレンダリングをおこないます。必要であれば、QTextEdit を直接使って独自のヘルプブラウザを作成することもできます。QTextEdit は HTML 3.2 のサブセットをサポートし、QStyleSheet クラスで作成するカスタムタグを利用することもできます。

5.2 GUIアプリケーションのサンプル

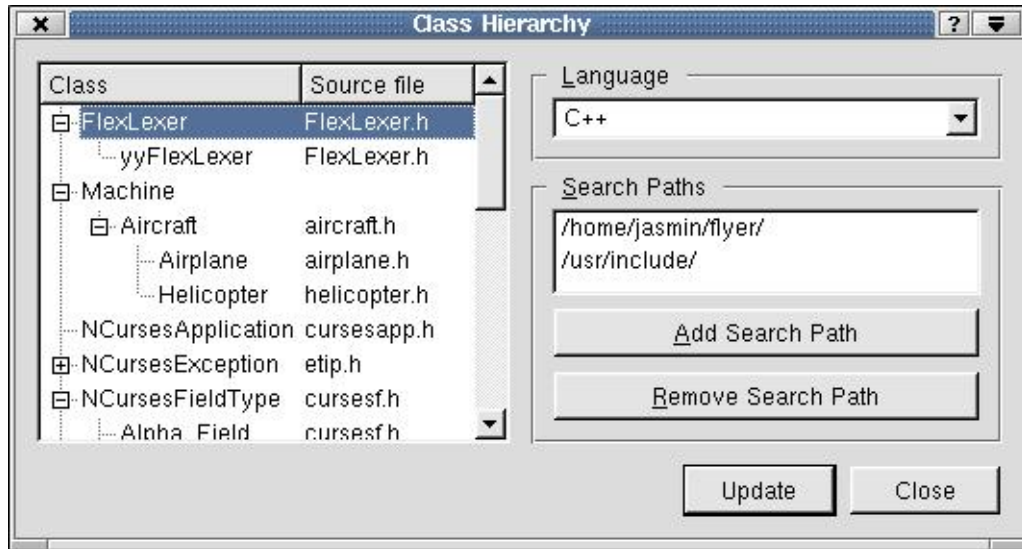


図 20. クラスビューワアプリケーション

「クラスビューワ」は伝統的なダイアログスタイルのアプリケーションで、いくつかのオプション（ここではパス）を選択できます。このオプション設定にしたがって処理がおこなわれます。

アプリケーションの完全なソースコードを以下に示します。main.cpp は Qt Designer によって生成されたものです。フォームは Qt Designer を使って作成し、*.ui ファイルに収められています。*.ui ファイルは uic を使って C++ ソースコードに変換されるため、プログラマはアプリケーションの機能面にだけ集中できます。

addSearchPath()、removeSearchPath()、updateHierarchy() は、すべてスロットです。スロットは、Qt Designer を使ってそれぞれ適切なボタンに接続されます。Qt Designer を使えばスロットの接続も視覚的におこなえます。

```
void ClassHierarchy::addSearchPath()
{
    QString path = QFileDialog::getExistingDirectory(
        QDir::homeDirPath(), this, 0, "Select a Directory" );
    if ( !path.isEmpty() &&
        searchPathBox->findItem(path, ExactMatch) == 0 )
        searchPathBox->insertItem( path );
}

void ClassHierarchy::removeSearchPath()
{
    searchPathBox->removeItem( searchPathBox->currentItem() );
}

void ClassHierarchy::updateHierarchy()
{
    QString fileNameFilter;
    QRegExp classDef;
    if ( language->currentText() == "C++" ) {
```

```

        fileNameFilter = "*.h";
        classDef.setPattern(
            "\\bclass\\s+([A-Z_a-z0-9]+)\\s*"
            "(?:\\{\\s*public\\s+([A-Z_a-z0-9]+))" );
    } else if ( language->currentText() == "Java" ) {
        fileNameFilter = "*.java";
        classDef.setPattern(
            "\\bclass\\s+([A-Z_a-z0-9]+)\\s+extends\\s*"
            "([A-Z_a-z0-9]+)" );
    }
    dict.clear();
    listView->clear();
    for ( int i = 0; i < searchPathBox->count(); i++ ) {
        QDir dir = searchPathBox->text( i );
        QStringList names = dir.entryList( fileNameFilter );
        for ( int j = 0; j < names.count(); j++ ) {
            QFile file( dir.filePath(names[j]) );
            if ( file.open(IO_ReadOnly) ) {
                QString content = file.readAll();
                int k = 0;
                while ( (k = classDef.search(content, k)) != -1 ) {
                    processClassDef( classDef.cap(1), classDef.cap(2),
                        names[j] );
                    k++;
                }
            }
        }
    }
}

void ClassHierarchy::processClassDef( const QString& derived,
    const QString& base, const QString& sourceFile )
{
    QListViewItem *derivedItem = insertClass( derived, sourceFile );
    if ( !base.isEmpty() ) {
        QListViewItem *baseItem = insertClass( base, "" );
        if ( derivedItem->parent() == 0 ) {
            listView->takeItem( derivedItem );
            baseItem->insertItem( derivedItem );
            derivedItem->setText( 1, sourceFile );
        }
    }
}

QListViewItem *ClassHierarchy::insertClass( const QString& name,
    const QString& sourceFile )
{
    if ( dict[name] == 0 ) {
        QListViewItem *item = new QListViewItem( listView, name,
            sourceFile );
        item->setOpen( true );
        dict.insert( name, item );
    }
    return dict[name];
}

```

オンラインリファレンス

<http://doc.trolltech.com/3.2/designer-manual.html>

6 2D/3D グラフィクス

Qt は 2D/3D グラフィクスを強力にサポートします。Qt の 2D グラフィクスクラスはビットマップタイプのグラフィックとベクタタイプのグラフィックに加え、アニメーション、図形の重なり検出をサポートします。また、非常に多くの種類の画像フォーマットの読み出しと保存もサポートされます。Unicode サポートのリッチテキスト、テキストの回転、せん断もサポートされます。Qt は、プラットフォームに依存しない OpenGL ツールキットのデファクトスタンダードでもあります。

6.1 2D グラフィクス

6.1.1 画像

BMP、GIF(*)、JPEG、MNG、PNG、PNM、XBM、XPM といったさまざまな画像フォーマットの入出力および処理は QImage クラスでおこないます。

Qt の組み込みウィジェット、たとえばボタン、ラベル、メニュー項目など、画像を表示できるものが多くあります。プッシュボタンにアイコンを表示するサンプルを示します。

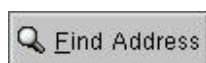


図 21. ボタンにアイコンを表示する

QImage クラスは 1 ビット、8 ビット、32 ビットの画像をサポートします。ピクセルデータ、パレットデータの処理、変形 (回転やせん断など)、減色処理 (必要があればディザリング) などをおこなうことができます。QImage に「アルファチャンネル」データを格納することもできるので、さまざまな用途 (透明処理やアルファブレンディング) に利用することができます。

QMovie は、アニメーション画像を表示することができるクラスです。

(*) 居住国でソフトウェア特許が認められ、LZW 圧縮法について Unisys 社がその国での特許を保有している場合、GIF 技術を使う場合 Unisys 社からライセンス使用を要求される可能性があります。当社の見解では、Unisys 社のこの特許は 2004 年いっぱい世界的に有効期限が切れます。

6.1.2 描画

QPainter はプラットフォームの API に依存しない描画用クラスです。標準的な描画機能に加え、変形やクリッピング処理といった高度な処理もサポートされます。Qt の組み込みウィジェットは、すべて自身自身の描画に QPainter を使います。カスタムウィジェットを実装する場合には、必ず QPainter を使う必要が生じます。

QPainter の標準的な機能を使い、点、線、多角形、楕円、円弧、ベジェ曲線などの描画をおこなえます。(25, 15) の位置に 120 × 60 の長方形を赤の点線で 2 ピクセルの線幅で描画するサンプルです。

```
painter.setPen( QPen(red, 2, DashLine) );  
painter.drawRect( 25, 15, 120, 60 );
```

デフォルトでは、ウィジェットの左上の位置は (0, 0) で、右下の位置が (width()-1, height()-1) となります。QPainter の座標系は変形、伸縮、剪断することができます。オブジェクトは「ウィンドウ」でクリッピングし、位置を「ビューポート」で指定します。

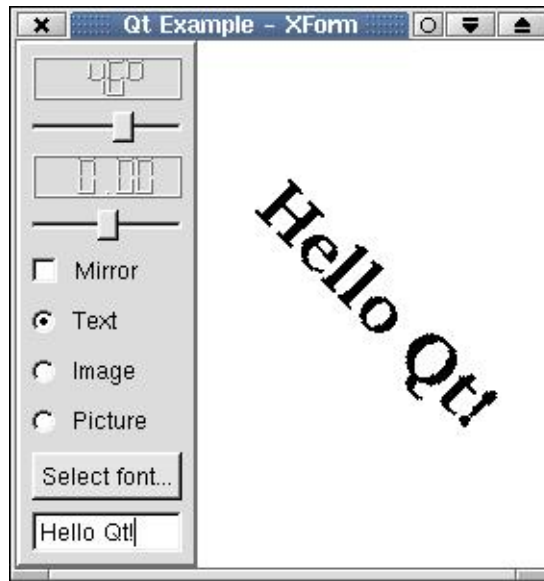


図 22. Qt に同梱されている xform サンプルプログラムでテキストを回転するボタンにアイコンを表示する

次のサンプルは棒グラフカスタムウィジェットの描画部分です。QPainter を使って paintEvent() をオーバーライドしています。座標系はデフォルトを使います。

```
void BarGraph::paintEvent( QPaintEvent * )
{
    QPainter painter( this );
    draw_bar( &painter, 0, 39, Qt::DiagCrossPattern );
    draw_bar( &painter, 1, 31, Qt::BDiagPattern );
    draw_bar( &painter, 2, 44, Qt::FDiagPattern );
    draw_bar( &painter, 3, 68, Qt::SolidPattern );
    painter.setPen( black );
    painter.drawLine( 0, 0, 0, height() - 1 );
    painter.drawLine( 0, height() - 1, width() - 1, height() - 1 );
    painter.setFont( QFont( "Helvetica", 18 ) );
    painter.drawText( rect(), AlignHCenter | AlignTop, "Sales" );
}

void BarGraph::draw_bar( QPainter *painter, int month, int barHeight,
    BrushStyle pattern )
{
    painter->setPen( blue );
    painter->setBrush( QBrush( darkGreen, pattern ) );
    painter->drawRect( 10 + 30 * month, height() - barHeight, 20,
        barHeight );
}
```

width()、height()、rect() 関数を使ってウィジェットのサイズを決定しているで、異なるサイズのウィジェットでも適切に描画されます。上記のコードの実行結果を図に示します。

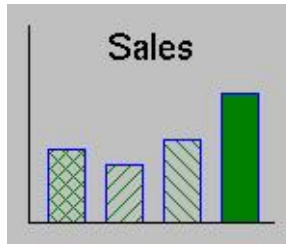


図 23. カスタムウィジェット

QPainter は、クリッピングをサポートしています。クリッピング領域は長方形、多角形、楕円、ビットマップで指定できます。単純なリージョンを結合、交差、差分、XOR などの処理で組み合わせることで複雑なリージョンを作成することもできます。クリッピングをおこなうことで、再描画の際のフリッカーを軽減することができます。

QColor は RGB、HSV、色名 ("skyblue" など) で指定した色を格納するクラスです。24 ビットカラーを指定することができます。Qt は指定された色、または色数が少ないシステムの場合似た色をシステムパレットから自動的に割り当てます。

6.1.3 描画デバイス

QPainter はどの種類の「描画デバイス」上にも描画することができます。サポートされている描画デバイスに描画するためのコードは、デバイスの種類にかかわらず同じ処理ですみます。Qt でサポートされている描画デバイスは次の通りです。

- QPixmap. QPixmap は「オフスクリーンウィジェット」として利用します。まず QPixmap に描画し、次に QWidget にビット転送するという処理をおこなうことでフリッカーを軽減することができます。これは「ダブルバッファリング」と呼ばれるプログラミング技法です。
- QPicture. QPicture はベクタ画像を格納するクラスです。ベクタ画像は自由にサイズ変更、回転、せん断することができます。QPicture クラスは、画像をピクセルデータではなく描画コマンドのリストとして格納します。QPicture クラスは SVG XML フォーマット (Scalable Vector Graphics, W3C が定義) で画像の読み出しと書き込みがおこなえます。
- QPrinter. QPrinter は物理プリンタをあらわすクラスです。Windows では、印刷コマンドは Windows の印刷エンジン (印刷エンジンはプリンタドライバを利用します) に直接送られます。Unix では、PostScript コマンドがプリントデーモンに送られます。
- QWidget. QWidget も描画デバイスです。QWidget を描画デバイスとして使う例は棒グラフカスタムウィジェットのサンプルで示されています。

6.1.4 キャンパス

QCanvas は 2D グラフィクスの高レベルインターフェイスです。キャンパスには、直線、長方形、楕円、テキスト、ピクスマップ、アニメーションスプライトなどの描画要素をきわめて多数持たせることがで

きます。描画要素は、ユーザが操作可能にする（移動するなど）ことが簡単におこなえます。描画要素をあらわすクラスは `QCanvasItem` のサブクラスです。`QCanvasItem` はウィジェットよりも軽量で、移動する、隠す、表示するなどを高速におこなえます。`QCanvas` は効率的な図形の重なり検知機能を備えているため、指定された領域に含まれる描画要素をすべてリストアップすることができます。`QCanvasItem` をサブクラス化してカスタムタイプを作成し、既存の描画要素に付加機能を追加することもできます。

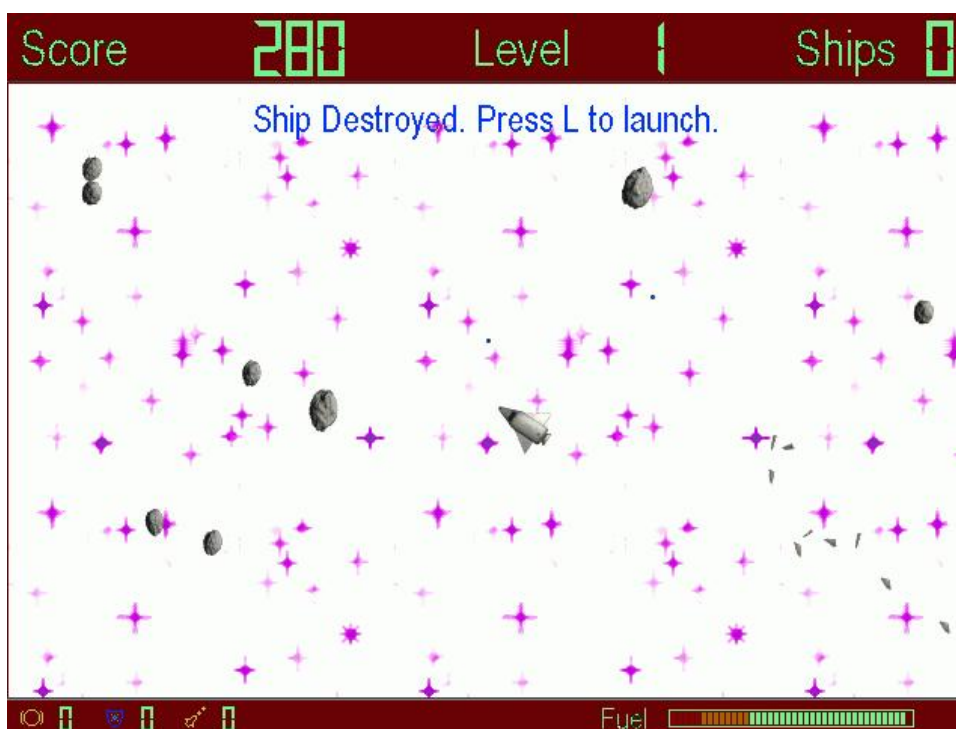


図 24. `QCanvas` を使って作成されたゲーム KAsteroids

`QCanvas` は `QCanvasView` クラスを使ってレンダリングされます。1 つの `QCanvas` を複数の `QCanvasView` を使い、異なる座標変換、サイズ、回転、せん断で表示することができます。

`QCanvas` はデータの視覚化に最適です。Qt を用いる開発者は、道路案内図を描いたりネットワーク構成を表示したりするのに `QCanvas` を使っています。また、スプライトを多数使う高速な 2D ゲームにも適しています。

6.2 3D グラフィクス

OpenGL(*) は 3D グラフィクスレンダリングの標準 API です。GUI アプリケーションで OpenGL を使って 3D グラフィクスを描画する際に Qt を利用する開発者も多くいます。`QWidget` のサブクラスである `QGLWidget` をサブクラス化し、`QPainter` ではなく OpenGL の標準関数を使って描画します。

Qt の OpenGL モジュールは Windows、X11、Mac OS X のすべてで利用可能で、それぞれのプラットフォーム標準の OpenGL (または Mesa) ライブラリを利用します。

OpenGL レンダリングコンテキストの表示フォーマットはシングル、ダブルバッファリング、デプスバッファ、RGBA またはカラーインデックスモード、アルファチャンネル、オーバーレイなどを使用することができます。カラーインデックスモードでは、独自のカラーマップを設定することもできます。

(*)OpenGL は Silicon Graphics 社の米国その他の登録商標です。

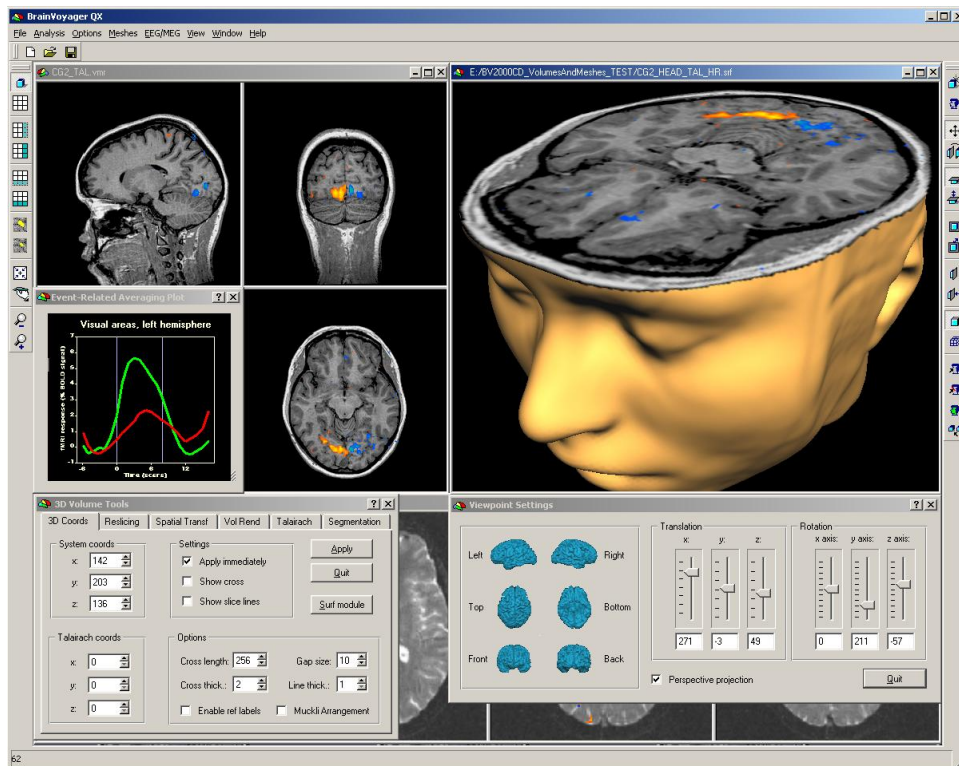


図 25. Brain Innovation 社の BrainVoyager は Qt と OpenGL で開発されています

Qt を使った場合でも、完全に OpenGL のコードを記述できます。Qt には `qglClearColor()`、`qglColor()` という 2 つの簡易関数が用意されています。どちらも、パラメータに `QColor` を指定することができ、どのモードでも利用できます。

6.3 3D グラフィクスのサンプル

3D の直方体を描画するアプリケーションの完全なサンプルコードを以下に示します。X、Y、Z 軸のスライダがついており、スライダを動かすことで直方体を回転させることができます。

`box3d.h` で `Box3D` クラスが宣言されています。

```
#include <qgl.h>
class Box3D : public QWidget
{
    Q_OBJECT
public:
    Box3D( QWidget* parent = 0, const char* name = 0 );
    ~Box3D();
public slots:
    void setRotationX( int deg ) { rotX = deg; updateGL(); }
    void setRotationY( int deg ) { rotY = deg; updateGL(); }
    void setRotationZ( int deg ) { rotZ = deg; updateGL(); }
protected:
```

```

    virtual void initializeGL();
    virtual void paintGL();
    virtual void resizeGL( int w, int h );
    virtual GLuint makeObject();
private:
    GLuint object;
    GLfloat rotX, rotY, rotZ;
};

```

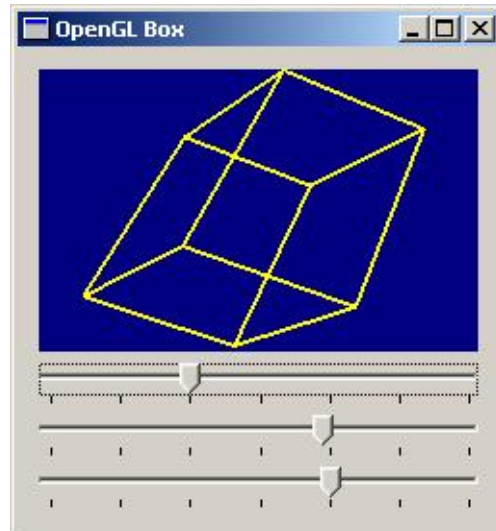


図 26. 3D 直方体

box3d.cpp で box3d.h で宣言されたクラスが定義されます。

```

#include "box3d.h"
Box3D::Box3D( QWidget* parent, const char* name )
: QWidget( parent, name )
{
    object = 0;
    rotX = rotY = rotZ = 0.0;
}

Box3D::~Box3D()
{
    makeCurrent();
    glDeleteLists( object, 1 );
}

void Box3D::initializeGL()
{
    qglClearColor( darkBlue );
    object = makeObject();
    glShadeModel( GL_FLAT );
}

void Box3D::paintGL()
{
    glClear( GL_COLOR_BUFFER_BIT );

```

```

        glLoadIdentity();
        glTranslatef( 0.0, 0.0, -10.0 );
        glRotatef( rotX, 1.0, 0.0, 0.0 );
        glRotatef( rotY, 0.0, 1.0, 0.0 );
        glRotatef( rotZ, 0.0, 0.0, 1.0 );
        glCallList( object );
    }

void Box3D::resizeGL( int w, int h )
{
    glViewport( 0, 0, w, h );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glFrustum( -1.0, 1.0, -1.0, 1.0, 5.0, 15.0 );
    glMatrixMode( GL_MODELVIEW );
}

GLuint Box3D::makeObject()
{
    GLuint list = glGenLists( 1 );
    glNewList( list, GL_COMPILE );
    glColor( yellow );
    glLineWidth( 2.0 );
    glBegin( GL_LINE_LOOP );
    glVertex3f( +1.5, +1.0, +0.8 );
    glVertex3f( +1.5, +1.0, -0.8 );
    /* ... */
    glEnd();
    glEndList();
    return list;
}

```

main.cpp では Box3D をインスタンス化し、3つのスライダを作成しています。

```

#include <qapplication.h>
#include <qslider.h>
#include <qvbox.h>
#include "box3d.h"
void create_slider( QWidget *parent, Box3D *box3d, const char *slot )
{
    QSlider* slider = new QSlider( 0, 360, 60, 0,
        QSlider::Horizontal, parent );
    slider->setTickmarks( QSlider::Below );
    QObject::connect( slider, SIGNAL(valueChanged(int)), box3d, slot );
}

int main( int argc, char** argv )
{
    QApplication::setColorSpec( QApplication::CustomColor );
    QApplication app( argc, argv );

    if ( !QGLFormat::hasOpenGL() )
        qFatal( "This system has no OpenGL support" );
    QVBox* parent = new QVBox;
    parent->setCaption( "OpenGL Box" );
    parent->setMargin( 11 );
    parent->setSpacing( 6 );
    Box3D* box3d = new Box3D( parent );
}

```

```
        create_slider( parent, box3d, SLOT(setRotationX(int)) );
        create_slider( parent, box3d, SLOT(setRotationY(int)) );
        create_slider( parent, box3d, SLOT(setRotationZ(int)) );
        app.setMainWidget( parent );
        parent->resize( 250, 250 );
        parent->show();

    return app.exec();
}
```

オンラインリファレンス

<http://doc.trolltech.com/3.2/coordsys.html>

<http://doc.trolltech.com/3.2/canvas.html>

<http://doc.trolltech.com/3.2/opengl.html>

7 データベース

Qt の SQL モジュールを使うと、マルチプラットフォームの GUI データベースアプリケーションを簡単に作成することができます。SQL 文を実行、特定のデータベースに対応したウィジェットの使用、任意のウィジェットのデータベース対応が可能です。

Qt の SQL モジュールは SQL データベースへのマルチプラットフォームのアクセスインターフェイスを備えています。Qt には Oracle、Microsoft SQL Server、Sybase Adaptive Server、IBM DB2、PostgreSQL、MySQL、ODBC のネイティブドライバが含まれています。データベースドライバは、Qt が対応するすべてのプラットフォームでサポートされ、クライアントライブラリが利用できます。複数のドライバを使えば、複数のデータベースに同時にアクセスできます。

任意の標準 SQL 文を簡単に実行できます。Qt は高レベルの C++ インターフェイスも持っているため、適切な SQL 文を自動的に生成することもできます。

Qt ウィジェットは、組み込みウィジェットでもカスタムウィジェットでも、簡単にデータベース対応にすることができます。また、データベース対応の簡便なウィジェットも用意されているので、データベースレコードをフォーム形式やテーブル形式で表示するダイアログやウィンドウを簡単に作成することができます。ウィジェットをデータベース対応にすると、データベースレコードの表示、更新、削除を自動的にサポートするようになります。テーブル定義によっては、新しく挿入するレコードは一意的なキーでなければならないこともあります。Qt はこのような制約を前提としていないため、レコード挿入の際にはちょっとしたコードを記述して自分で制約処理をおこなう必要があります。操作（たとえば削除など）の前にユーザに確認することも簡単です。

Qt の SQL モジュールは Qt Designer と完全に統合されています。Qt Designer にはテンプレートとウィザードが用意されているので、データベースフォームを最小限の労力で作成することができます。ウィザードを使うと、ナビゲーション、更新、挿入、削除の各ボタンが配置されたフォームを作成できます。

Qt の SQL モジュールの機能を利用すれば、外部キー参照、マスターテーブル関係図、ドリルダウンなどをサポートするデータベースアプリケーションの作成は非常に容易になります。

7.1 SQL コマンドの実行

QSqlQuery クラスは SQL 文を直接実行する際に使います。QSqlQuery クラスは、SELECT 文のリザルトセットを参照するためにも使われます。

クエリを実行し、QSqlQuery::next() を使ってリザルトセットを得るサンプルです。

```
QSqlQuery query( "SELECT id, surname FROM staff" );
while ( query.next() ) {
    cout << "id: " << query.value( 0 ).toInt()
    << " surname: " << query.value( 1 ).toString() << endl;
}
```

結果の値は SELECT 文でフィールドが出現した順に並べられます。QSqlQuery には、first()、prev()、last()、seek() といったリザルトセット内を移動する関数も用意されています。

INSERT、UPDATE、DELETE などのコマンドの発行も同様に簡単です。UPDATE 文のサンプルを以下に示します。

```
QSqlQuery query( "UPDATE staff SET salary = salary * 1.10"
    " WHERE id > 1155 AND id < 8155" );
if ( query.isActive() ) {
    cout << "Pay rise given to " << query.numRowsAffected()
```

```

    << " staff" << endl;
}

```

Qt の SQL モジュールは、値バインド、プリペアドクエリといった動的 SQL をサポートします。

```

QSqlQuery query;
query.prepare( "INSERT INTO staff (id, surname, salary)"
    " VALUES (:id, :surname, :salary)"
query.bindValue( ":id", 8120 );
query.bindValue( ":surname", "Bean" );
query.bindValue( ":salary", 29960.5 );
query.exec();

```

上のサンプルで示されるように、値バインドは名前バインドおよび名前付きプレースホルダを利用するか、次のように番号バインドおよび番号プレースホルダを利用します。

```

QSqlQuery query;
query.prepare( "INSERT INTO staff (id, surname, salary)"
    " VALUES (?, ?, ?)"
EmployeeMap::iterator it;
for ( it = employeeMap.begin(); it != employeeMap.end(); ++it ) {
    query.addBindValue( it.data().id() );
    query.addBindValue( it.key() );
    query.addBindValue( it.data().salary() );
    query.exec();
}

```

Qt のバインド構文は、データベースエンジンに直接発行するか、エミュレーションをおこなうことで、サポートされているすべてのデータベースで使うことができます。SQL 文を直接記述したくない場合は、SQL テーブルやビューのレコードを表示したり編集する高レベルインターフェイスである QSqlCursor を使います。QSqlCursor を使えば、SQL 文を記述する必要はなくなります。

```

QSqlCursor cur( "staff" );
while ( cur.next() ) {
    cout << "id: " << cur.value( "id" ).toInt()
    << " surname: " << cur.value( "surname" ).toString() << endl;
}

```

QSqlCursor は、SQL 文の ORDER BY 句と WHERE 句に相当するソートとフィルタリングもサポートしています。

演算フィールドは実際の計算（合計値を計算するなど）と外部キー検索（コードではなく名前で表示するなど）の両方の用途に使えます。演算フィールドを作成するには、QSqlCursor をサブクラス化し、QSqlCursor::calculateField() をオーバーライドし、演算結果を格納する QSqlField に true をセットします。

データベースドライバのほとんどは、フィールドの実際のデータ型によらず値を文字列型として扱うことができます。Qt は、このようなデータの扱いを QVariant クラスを使うことでシームレスにおこないます。データベースドライバがこの機能を持っているかを問い合わせることができます。また、あわせてレポートとトランザクションのクエリサイズも返されます。データベースがトランザクション機能をサポートする場合、transaction()、commit()、rollback() の各関数も利用できます。

7.2 データベースウィジェット

QDataTable は QSqlCursor を使ってリザルトセットのレコードを表示できる QTable です。QDataTable は QTable と同様、セルを直接編集できます。QDataTable の確認プロパティを設定することで、すべての変更、または変更の一部を実行する前にユーザに確認を求めることができます (削除など)。編集に使われるウィジェットは、各データタイプに応じて自動的に選択されます。たとえば CHAR 型のフィールドには QLineEdit が、INTEGER 型のフィールドには QSpinBox が使われるなどです。そのテーブルのプロパティマップを作成することで、デフォルトの動作を変更することができます。プロパティマップは、フィールド (カラム) とエディタウィジェットとを関連付けるものです。

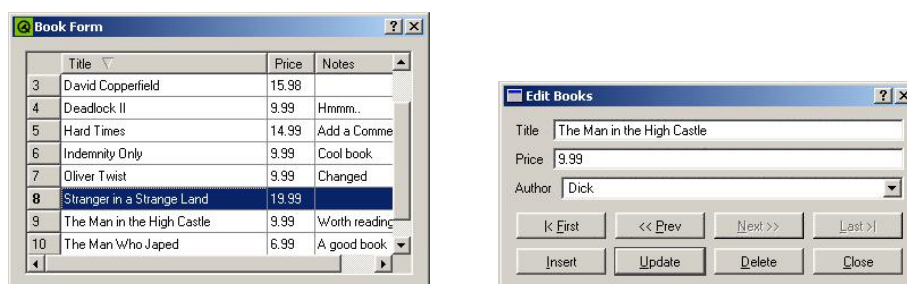


図 27. QDataTable と QDataBrowser

レコードの更新と削除はいっさいコードを記述せずおこなえます。挿入は若干のコードを記述する必要がありますが、レコードがユニークなキーを持たなければならないテーブルがほとんどだからです。これは、QDataTable::beforeInsert() シグナルにスロットを接続し、キーを生成するコードを記述することでごく簡単におこなうことができます。

QDataTable は巨大なリザルトセットを高速に読み込んで、ユーザへの応答を即座におこなえるように、バッファリングをおこないます。クエリの結果サイズを返す機能を持つデータベースエンジンの場合には、即座にスクロールバーのスライダを適切なサイズで表示することもできます。

Qt には QDataBrowser と QDataView というクラスも用意されています。どちらも少数のレコードを表示するのに適したフォームウィジェットです。レコードを移動するナビゲートボタンがあらかじめ用意されています。QDataView はリードオンリーのデータ表示に、QDataBrowser はすでに挿入、更新、削除の各ボタンが用意されているためレコードの編集に適しています。

QDataTable と QDataBrowser はどちらもレコードの編集のためのコンテキストメニューとキーボードショートカットを持っています。

primeInsert() シグナルと primeUpdate() シグナルを接続したスロットを作成することで、データベースから取得したレコードデータを、表示する前に何らかの処理をおこなうことができます。

また同じように、レコードの変更をデータベースに書き戻す前に、データの処理をおこなったり動作のログを取ることもできます。たとえば外部キーの表示用文字列を ID に戻すなどです。これは、beforeInsert() シグナル、beforeUpdate() シグナル、beforeDelete() シグナルにスロットを接続することでおこなえます。

データベースレコードを表示する独自のフォームを作成することもできます。従来のツールキットのように、同じような名前のデータベース対応版のウィジェットは Qt にはありません。Qt のウィジェット (カスタムウィジェットも含めて) は最初からデータベース対応になっています。ウィジェットを QSqlForm に持たせ、データベースフィールドと、フィールドの編集に使うためのウィジェットを対応づけるプロパティマップを設定するだけです。

テーブルのマスター/明細関係も簡単に設定することができます。明細フォームやカーソル行のレコードを、マスターフォームまたはマスタテーブルのカレントレコードを使ってフィルタリングすればよいだけです。データのドリルダウンも同じように簡単です。ボタン、メニュー項目、キーボードショートカットなどをドリルダウンフォームに結びつけ、レコードのキーをパラメータとしてドリルダウンフォームを表示すればよいだけです。

Qt Designer は、Qt の SQL モジュールを完全に統合しています。Qt Designer は、データベースの現在のデータをプレビューすることができ、必要であればレコードの表示、削除、更新もおこなえます。Qt Designer は、データベースフォームを素早く簡単に作成できるようなテンプレートとウィザードを備えています。

オンラインリファレンス

<http://doc.trolltech.com/3.2/sql.html>

8 国際化

Qt は国際標準文字集合 Unicode をサポートしています。Unicode を採用することで、適切なフォントを用いれば、アプリケーションでアラビア語、英語、ヘブライ語、日本語、ロシア語を含むさまざまな言語を問題なく混在させることができます。Qt はアプリケーションの翻訳を支援するツールも含まれているため、自分たちのアプリケーションを国際市場に進出させることもできるのです。

Qt は、翻訳を支援するツールを持っています。ユーザの目に触れるため、翻訳が必要なテキストをマークしておくことで、ソースコードから翻訳が必要なテキストを抽出するツールです。Qt Linguist は操作の容易な GUI アプリケーションで、ソースコードからテキストを抽出し、補助情報とともに表示することで翻訳を容易におこなえます。Qt Linguist が出力するドキュメントは、リリース管理者、翻訳者、プログラマに適切な情報として役立てることができます。

8.1 Unicode

Qt は Unicode 文字列の格納に QString クラスを使っています。Qt の API や内部処理すべてにわたって QString が使われます。従来の `const char*` 型は QString に、`char` 型は 16 ビット長の QChar に置き換えられます。コンストラクタや演算子関数では、自動的に 8 ビット文字列から QString に変換がおこなわれます。QString は値のコピーがなされるので、内部的に同じ文字列を共有します (コピーオンライト)。このため、コピーは高速でメモリの利用効率が高まります。

QString は単に文字列を 16 ビット化しただけではありません。QChar::lower() や QChar::isPunct() といったメソッドが用意されており、C 言語標準ライブラリの tolower() や ispunct() といった関数を使う必要はありません。これらのメソッドはすべて Unicode に対応しています。Qt の正規表現エンジンは QRegExp クラスで実装されています。QRegExp クラスでは、正規表現文字列にも対象文字列にも Unicode を使うことができます。

エンコードと文字セットの変換は QTextCodec サブクラスでおこないます。QTextCodec はフォント、I/O、インプットメソッドなど、あらゆる文字列処理で使われます。また、プログラマが自由に利用することもできます。

Qt 3.2 では 38 種類のエンコードをサポートしています。中国語の Big5 と GBK、日本語の EUC-JP、JIS、Shift-JIS、ロシア語の KOI8-R、ISO 8859 系などです。<http://doc.trolltech.com/3.2/qtextcodec.html> で、すべてのエンコードの一覧を見ることができます。キャラクタマップを作成するか、QTextCodec をサブクラス化することで、独自のエンコードを作成することもできます。

8.2 テキスト入力とレンダリング

アジア圏の言語は、キーボードで利用できる文字よりもはるかに多い文字を使います。あるキーの組み合わせから実際の文字に変換する処理は、ウィンドウシステムレベルのプログラムでおこなわれます。これは「インプットメソッド」と呼ばれます。Qt は、インストール済みのインプットメソッドを自動的に検出し、利用できるようにします。

Qt は強力なテキストレンダリングを備えており、シンプルなラベルから洗練されたリッチテキストエディタまで、スクリーン上に表示するすべてのテキストのレンダリングをおこないます。レンダリングエンジンは特殊な改行、右から書く言語、読み分け記号などもサポートします。また、アラビア語、中国語、キリル語、英語、ギリシャ語、ヘブライ語、日本語、韓国語、ベトナム語など、世界中の言語をサポートします。Qt は複数の言語からなる文字列のフォントを適切に判断し、自動的に組み合わせるとのレン

ダリングをおこないます。

8.3 アプリケーションの翻訳

Qt には、アプリケーションユーザの母国語に翻訳するツールと関数が用意されています。文字列を翻訳用にマークするには、文字列を `tr()` 関数 ("translation" の略です) で囲むだけです。

```
saveButton->setText( tr("Save") );
```

`tr()` は、文字列 (たとえば "Save") を翻訳文字列がある場合、それに置換しようとしています。翻訳文字列がない場合は元の文字列がそのまま使われます。たとえば英語を元の言語に、中国語を翻訳後の言語にする、逆に中国語を元の言語に、英語を翻訳後の言語にすることができます。`tr()` の引数はアプリケーションのデフォルトエンコードから Unicode に変換されます。

`tr()` の一般的な構文は次の通りです。

```
Context::tr("source text", "comment")
```

"Context" は `QObject` のサブクラスのいずれかをあらわします。これは省略されることが多く、その場合 `tr()` メソッドを持っているクラスがコンテキストとして使われます。"source text" が翻訳する文字列で、"comment" はオプションです。"comment" は、コンテキスト以外に翻訳者に伝えたい情報を渡します。

翻訳結果は `QTranslator` オブジェクトに格納され、.qm ("Qt Message" ファイル) という拡張子のファイルに保存されます。それぞれの .qm ファイルには特定の言語に翻訳した結果が収められます。言語はロケールの設定やユーザの指定に応じて実行時に選択されます。

Qt には、.qm ファイルを扱うツールが 3 つ用意されています。lupdate、lrelease、それに Qt Linguist です。

1. lupdate はソースコード (Qt Designer の出力する .ui ファイルも含む) から「コンテキスト、ソース文字列、コメント」の 3 つ組を抽出し、.ts ("Translation Source") ファイルを生成します。.ts ファイルは XML フォーマットなので、人間が読める形式です。
2. 翻訳者が Qt Linguist を使って .ts ファイルに納められているソース文字列の翻訳をおこないます。
3. .ts ファイルを lrelease で処理し、.qm ファイルを出力します。.qm ファイルは圧縮されています。

この作業はアプリケーションのライフサイクル全体にわたって繰り返しおこなわれます。lupdate は、何度実行してもまったく安全です。lupdate は古いソース文字列に対する翻訳済みの部分を削除するのではなく、「翻訳済み」のマークをつけて残すからです。また、ソース文字列の小さな変更を検出し、適切な翻訳結果を自動的に候補として表示します。小さな変更が加えられたソース文字列は未翻訳状態になるため、翻訳者のチェックも容易になります。

Qt には、あらかじめよく使われる文字列の翻訳が 400 ほど用意されています。 TrollTech 社が翻訳したフランス語とドイツ語があります。

8.4 Qt Linguist

Qt Linguist は Qt アプリケーションの翻訳を支援するツールです。

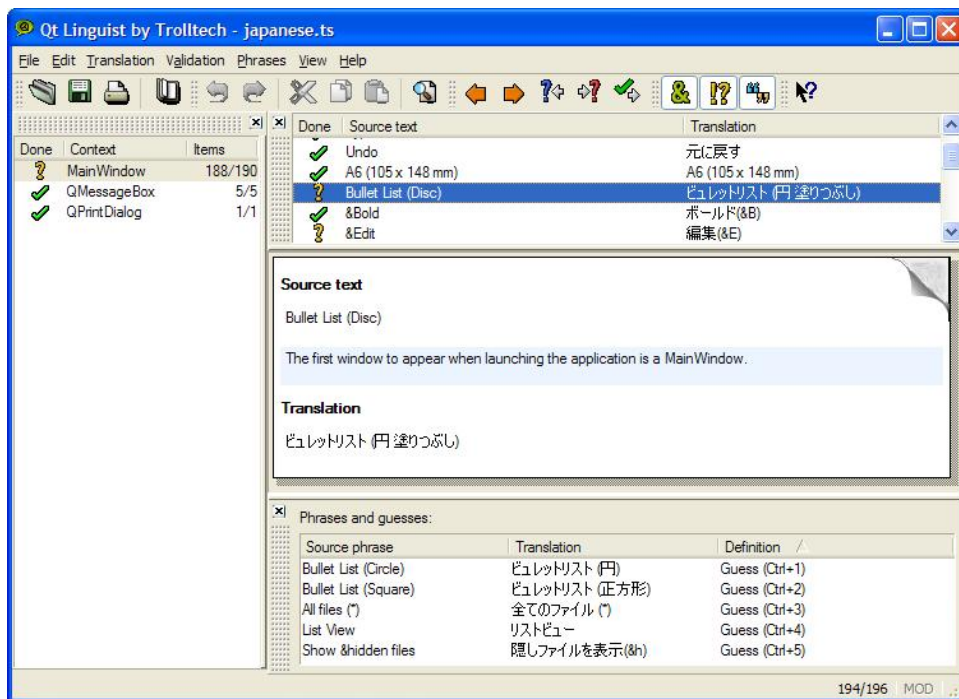


図 28. Qt Linguist

Qt Linguist を使えば、.ts ファイルの翻訳が簡単におこなえます。左のウィンドウに .ts ファイルのコンテキストが表示され、右上のウィンドウには現在のコンテキストのソース文字列が翻訳結果とともに表示されます。ソース文字列を選択して翻訳をおこない、翻訳完了か未翻訳のマークをつけて次の未翻訳文字列に移動します。Done & Next、Next Unfinished といったよく使う移動機能には、すべてキーボードショートカットが割り当てられています。ウィンドウはドッキング可能なので、翻訳者の好みにあわせてカスタマイズすることができます。

1 つのアプリケーションの中で、同じ言い回しを何度も使うことがあります。同じ文字列でも別々のソース文字列となっていることが多いのですが、Qt Linguist は 以前の翻訳結果を自動的に判断し、候補として下のウィンドウに表示します。こうした候補は似たような言い回しの表現を統一するための手助けとなります。Qt Linguist は、キーボードアクセラレータとサブメニューを持つメニュー項目に表示される”...”が正しく翻訳されているかをチェックするオプションもあります。

オンラインリファレンス

<http://doc.trolltech.com/3.2/i18n.html>

<http://doc.trolltech.com/3.2/unicode.html>

<http://doc.trolltech.com/3.2/scripts.html>

<http://doc.trolltech.com/3.2/linguist-manual.html>

9 スタイルとテーマ

Qt では、何も指定がなければネイティブスタイルのルック&フィールを使います。Qt アプリケーションは色やフォント、サウンドといったユーザの好みを尊重しますが、他のスタイルを使ってユーザの設定を変更してはいけないということではありません。Qt の強力なスタイルエンジンを使えば、既存のスタイルを変更したり、独自のスタイルを作成することもできます。

スタイルは、特定のプラットフォームでユーザインターフェイスのルック&フィールを設定するためのものです。スタイルは `QStyle` をサブクラス化し、直線を描画する、ボタンを描画するといった基本的な描画関数を再実装することで実現されています。Qt はウィジェットの描画を可能な限り高速に、かつ柔軟におこないます。

9.1 組み込みスタイル

Qt には、Windows、Windows XP、Motif、MotifPlus、CDE、Platinum、SGI、Mac という組み込みスタイルが用意されています。デフォルトではプラットフォームやデスクトップ環境にあったスタイルを使いますが、スタイルをプログラマ的に変更することも、`-style` コマンドラインオプションで変更することもできます。スタイルを補完するものがテーマです。テーマはユーザの好みのフォント、色、サウンドなどをまとめたものです。



図 29. 組み込みスタイルでそれぞれコンボボックスを表示する

Qt は、実行環境でのアクティブなテーマを自動的に選択します。たとえば、Windows 環境ではメニューやツールチップのアニメーションがおこなわれます。

Windows XP スタイルと Mac スタイルだけはネイティブのスタイルマネージャを使ってスタイル処理がおこなわれているため、ネイティブプラットフォーム上でしか利用できません。しかし、その他のスタイルは Qt がエミュレートするので、どのプラットフォームでも使うことができます。

9.2 スタイル対応ウィジェット

Qt の組み込みウィジェットはどれもスタイルに対応しています。カスタムウィジェットやカスタムダイアログは、ほとんどの場合組み込みウィジェットとレイアウトを組み合わせで作成されるので、自動的にスタイル対応となります。ごくまれにまったく新たにカスタムウィジェットを作成する必要がある場合もありますが、そういった場合には自前で長方形を描画するのではなく、`QStyle` を使って基本的なユーザインターフェイスを描画します。

9.3 スタイルのカスタマイズ

アプリケーションまたはアプリケーション群のルック&フィールを独自のものにしたい場合には、カスタムスタイルを使います。カスタムスタイルは `QStyle` が `QCommonStyle`、または `QCommonStyle` のサ

ブクラスのいずれかを継承することで作成できます。既存のスタイルを変更することは簡単で、適切な基本クラスの1つか2つの仮想関数を再実装すればよいだけです。

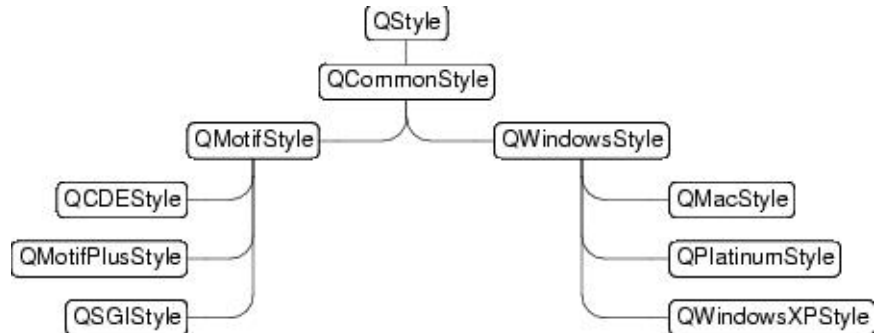


図 30. QStyle クラス階層

アプリケーションのスタイルは次のように設定します。

```
QApplication::setStyle( new MyCustomStyle );
```

スタイルはプラグインとして作成することもできます。スタイルをプラグインとして作成しておけば、Qt Designer でカスタムスタイルを使ってフォームのプレビューをおこなうことができます。そうすれば、Qt や Qt Designer の再コンパイルは必要ありません。また、Qt アプリケーションのスタイル変更も再コンパイルせずおこなうことができます。

オンラインリファレンス

<http://doc.trolltech.com/3.2/customstyle.html>

10 レイアウト

レイアウトを使えば、サイズや位置を固定することなく、柔軟な表示のフォームを作成することができます。レイアウトを使うことで、プログラマはサイズや位置の調整から解放されます。実行時の画面サイズや言語、フォントにあわせて自動的に表示が調整されます。

Qt は、子ウィジェットを親ウィジェットに配置するレイアウトマネージャを備えています。子ウィジェットの位置やサイズを自動的に調整します。トップレベルウィジェットのデフォルトサイズや最小サイズ、内容やフォントが変更されたときに自動的に移動・リサイズがおこなわれます。Qt Designer はウィジェットのレイアウトにレイアウトマネージャを使うように設計されています。

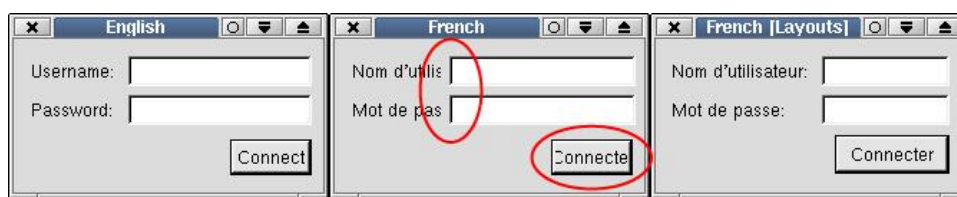


図 31. 英語、フランス語、およびレイアウト済みのフランス語

レイアウトは、アプリケーションの国際化にも役立ちます。ウィジェットのサイズと位置が固定されると、翻訳後の文字列は収まりきれずに途中で切れてしまうことがよくあります。しかし、レイアウトを使えば子ウィジェットのサイズが自動的に調整されるのです。

10.1 組み込みレイアウトマネージャ

組み込みレイアウトマネージャとして `QHBoxLayout`、`QVBoxLayout`、`QGridLayout` が用意されています。

`QHBoxLayout` は、ウィジェットを水平方向に左から右に 1 列に並べるマネージャです。`QVBoxLayout` はウィジェットを垂直方向に上から下に 1 列に並べるマネージャです。`QGridLayout` はウィジェットを格子状に並べるマネージャです。ウィジェットは複数のセルにまたがることもできます。

Qt のレイアウトマネージャは、ほとんどの場合においてウィジェットのサイズを最適なものに設定するため、ウィンドウのリサイズはスムーズにおこなわれます。デフォルトの動作を変更したい場合、以下の方法を使うことでレイアウト処理を細かく調整することができます。

1. 子ウィジェットの最小サイズ、最大サイズまたは固定サイズを設定します。
2. ストレッチウィジェットまたはスペーサウィジェットを追加します。ストレッチウィジェットとスペーサウィジェットは正確にはウィジェットではありませんが、レイアウトの空の領域のサイズにフィットするものです。
3. 子ウィジェットのサイズポリシーを変更します。`QWidget::setSizePolicy()` を呼び出すことで子ウィジェットのリサイズの際の動作を調整することができます。子ウィジェットのサイズを大きくしたり、小さくしたり、固定したりすることができます。
4. 子ウィジェットのサイズヒントを変更します。`QWidget::sizeHint()` と `QWidget::minimumSizeHint()` は、それぞれウィジェットの表示内容に応じた最適なサイズ、および最小サイズを返します。組み込みウィジェットはこの 2 つのメソッドは適切に実装されています。

- 伸張係数を設定します。伸張係数は子ウィジェットのサイズ拡大を調整する値です。たとえばサイズ変更の際、ウィジェット A に $2/3$ のスペースを、ウィジェット B に $1/3$ のスペースを割り当てるなどです。

ウィジェット同士の「スペース」と、あるウィジェットの周囲に取られる「マージン」はレイアウトマネージャ全体に適用される値で、プログラムから設定可能です。Qt Designer は、デフォルトでは環境に応じた標準的な値を使います。

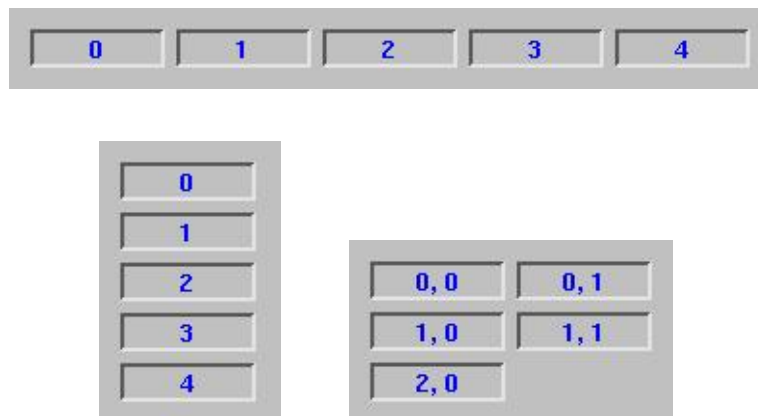


図 32. QHBoxLayout、QVBoxLayout、QGridLayout

レイアウトは右から左、下から上の順序に変更することもできます。右から左へのレイアウトは、右から書く言語（アラビア語やヘブライ語）をサポートする国際化アプリケーションにも役立ちます。

10.2 レイアウトのネスト

レイアウトはいくらでもネストすることができます。同じダイアログボックスを異なるサイズで表示した例を下の図に示します。



図 33. 小さなダイアログと大きなダイアログ

このダイアログは、3つのレイアウトマネージャを使っています。QVBoxLayout が3つのプッシュボタンをグループ化します。QHBoxLayout がプッシュボタンのグループと国リストをグループ化します。QVBoxLayout が「Select a country」のラベルと、残りのウィジェットをグループ化します。ストレッチウィジェットを使って「Cancel」ボタンと「Help」ボタンの距離を調整します。

このダイアログを作成するコードサンプルを示します。

```
QVBoxLayout *buttonBox = new QVBoxLayout( 6 );
buttonBox->addWidget( new QPushButton("OK", this) );
buttonBox->addWidget( new QPushButton("Cancel", this) );
buttonBox->addStretch( 1 );
buttonBox->addWidget( new QPushButton("Help", this) );
QListBox *countryList = new QListBox( this );
countryList->insertItem( "Canada" );
/* ... */
countryList->insertItem( "United States of America" );
QHBoxLayout *middleBox = new QHBoxLayout( 11 );
middleBox->addWidget( countryList );
middleBox->addLayout( buttonBox );
QVBoxLayout *topLevelBox = new QVBoxLayout( this, 6, 11 );
topLevelBox->addWidget( new QLabel("Select a country", this) );
topLevelBox->addLayout( middleBox );
```

このように、Qt のレイアウト機構は非常に使いやすいため、固定的な位置を指定する必要はほとんどありません。



図 34. Qt Designer でレイアウトを利用する

Qt Designer を使えば、レイアウトはもっと簡単になります。上のダイアログボックスの例で言えば、たった 17 回マウスをクリックしただけでウィジェットの作成とレイアウトが出来上がります。

10.3 レイアウトのカスタマイズ

QLayout をサブクラス化することでカスタムレイアウトマネージャを作成できます。Qt のパッケージに同梱されている customLayout サンプルは 3 つのカスタムレイアウトマネージャのサンプルです。

BorderLayout、CardLayout、SimpleFlow です。これらのサンプルを改変するところから始めてもよいでしょう。

Qt には、ユーザが移動できるスプリットバーをあらわす QSplitter クラスも用意されています。デザインによっては、レイアウトマネージャを使うより QSplitter を使う方が好ましい場合もあるでしょう。

さらに細かい制御をおこないたいければ、子ウィジェットの QWidget::resizeEvent() を再実装し、QWidget::setGeometry() を呼び出すことで、手動でレイアウトをおこなうこともできます。

オンラインリファレンス

<http://doc.trolltech.com/3.2/layout.html>

<http://doc.trolltech.com/3.2/customlayout.html>

10.4 イベント

アプリケーションオブジェクトは、システムメッセージを Qt のイベントとして受け取ります。アプリケーションはイベントをさまざまなレベルで監視したり、フィルタリングしたり、応答することができます。

Qt では、イベントは `QEvent` の継承クラスとしてあらわされます。イベントは `QObject` に通知されるため、`QObject` はイベントに応答することができます。イベントの監視やフィルタリングは、アプリケーションレベルでもオブジェクトレベルでもおこなえます。

10.5 イベントの作成

ほとんどのイベントはウィンドウシステムが生成し、ウィジェットに通知されます。たとえばキーの押下、マウスボタンのクリック、アプリケーションウィンドウのリサイズなどです。プログラマ的に生成したイベントを通知することも可能です。イベントの種類は 50 以上あります。主なイベントには `MouseButtonPress`、`MouseButtonRelease`、`MouseButtonDblClick`、`Wheel`、`KeyPress`、`KeyRelease`、`Paint`、`Resize`、`Close` などがあります。独自のイベントを作成することもでき、独自のイベントは組み込みのイベントとまったく同様に扱うことができます。

キーが押されたとか、マウスボタンが離されたというだけでは不十分な場合もあります。イベントの受信側はどのキーが押されたのか、どのマウスボタンが離されたのか、マウスポインタはどの位置にあったのかななどの付加情報も知る必要があります。こうした情報は `QEvent` のサブクラスである `QMouseEvent`、`QKeyEvent`、`QPaintEvent`、`QResizeEvent`、`QCloseEvent` で利用できます。

10.6 イベントの送信

イベントの通知は仮想関数 `QObject::event()` を呼び出すことでおこなわれます。よく使われるイベントは `QWidget::event()` からそのイベント専用のハンドラにそのまま転送されます。たとえば `QWidget::mousePressEvent()`、`QWidget::keyPressEvent()` などです。独自のウィジェットを作成した場合や、既存のウィジェットを変更した場合にはこれらのハンドラを再実装することもできます。

イベントには、即座に送信されるものといったキューに入れられて、Qt カーネルに制御が戻ってからディスパッチされるものがあります。キューを利用するのは、ある種のイベントを最適化するためです。たとえば、フリッカを最小限にし、処理を高速にするために、複数のペイントイベントを 1 つのイベントに変換するといった処理です。

あるオブジェクトのイベントを別のオブジェクトから監視したい場合もあります。つまり、イベントに応答するか、イベントの受け取りを拒否するかです。監視対象のオブジェクトの `QObject::installEventFilter()` を呼び出すことでおこないます。イベントが届くたびに監視側のオブジェクトの仮想関数 `QObject::eventFilter()` が呼び出されます。これにより、イベントはまず監視側のオブジェクトに渡され、監視対象のオブジェクトにはその後渡されます。

`qApp` にフィルタを登録することで、すべてのアプリケーションイベントをフィルタリングすることも可能です。`qApp` は `QApplication` の唯一のインスタンスです。アプリケーションレベルで登録されたフィルタは、ウィジェットごとのフィルタを呼び出す前に呼び出されます。イベントディスパッチャである `QApplication::notify()` を再実装すれば、イベントの処理を完全に制御することもできます。

オンラインリファレンス

<http://doc.trolltech.com/3.2/eventsandfilters.html>

<http://doc.trolltech.com/3.2/qapplication.html#notify>

11 入出力とネットワーク

Qt は、プレーンテキスト、XML ファイル、バイナリフォーマットの読み出しおよび書き込みがおこなえます。ローカルファイルの処理は専用のクラスを使い、リモートファイルは FTP プロトコルや HTTP プロトコルを使って処理します。プロセス間通信とソケットベースの TCP および UDP 通信も完全にサポートしています。

11.1 ファイル I/O

Qt はマルチプラットフォームの高度な I/O クラスを備えています。QTextStream クラスは C++ 標準ライブラリの <iostream> とほぼ同じインターフェイスを持ち、QTextCodec で扱われるエンコードを完全にサポートします。QDataStream クラスは、C++ の基本型と多くの Qt のクラスを、プラットフォームに依存しないバイナリ形式でシリアライズするために使われます。次のサンプルコードは Unicode 文字列、フォント、色を splash.dat というファイルに保存します。

```
QFile file( "splash.dat" );
if ( file.open(IO_WriteOnly) ) {
    QDataStream out( &file );
    out << QString( "SplashWidgetStyle" )
        << QFont( "Times", 18, QFont::Bold )
        << QColor( "skyblue" );
}
```

ファイルに保存したデータは次のようにして簡単に取り出すことができます。

```
QString str;
QFont font;
QColor color;
QFile file( "splash.dat" );
if ( file.open(IO_ReadOnly) ) {
    QDataStream in( &file );
    in >> str >> font >> color;
    if ( str == "SplashWidgetStyle" ) {
        splashWidget->setFont( font );
        splashWidget->setColor( color );
    }
}
```

QTextStream と QDataStream は QIODevice のどのサブクラスでも使うことができます。QIODevice のサブクラスには QFile、QBuffer、QSocket、QSocketDevice などが用意されており、独自の I/O デバイスを作成することもできます。QIODevice は readLine() や writeBlock() といった低レベルな関数も用意されており、ストリームの種類に限らず利用することができます。ディレクトリの読み出しや走査は QDir クラスでおこないます。QDir はパス名の処理やファイルシステムへのアクセス (ディレクトリの作成やファイルの削除など) のために使います。ファイルサイズ、パーミッション、作成日時、更新日時といった詳細な情報は QFileInfo で扱います。

ユーザのホームディレクトリにある隠しファイルをリストし、サイズの降順で表示するサンプルを示します。

```
QDir dir = QDir::home();
dir.setFilter( QDir::Files | QDir::Hidden );
dir.setSorting( QDir::Size | QDir::Reversed );
```

```

QStringList names = dir.entryList();
for ( int i = 0; i < names.count(); i++ ) {
    QFileInfo info( dir, names[i] );
    cout << names[i].latin1() << " " << info.size() << endl;
}

```

リモートファイルへの透過的なアクセスは `QUrlOperator` を使います。Qt はローカルファイルシステムへのアクセスだけでなく、FTP および HTTP プロトコルを使ったアクセスをサポートし、他のプロトコルをサポートするよう拡張することもできます。FTP を使ってファイルをダウンロードするには、次のようにします。

```

QUrlOperator op;
op.copy( "ftp://ftp.trolltech.com/qt/INSTALL", "file:/tmp" );

```

URL のパースと再構成は `QUrl` を使えば簡単におこなうことができます。

画像ファイルは `QImage` にファイル名を渡すことで読み込みおよび作成ができます。テキストと画像の印刷は `QPainter` を使います。これらのクラスについては「6.1. 2D グラフィクス」で詳しく解説されています。

11.2 XML

Qt の XML モジュールには、SAX パーサと DOM パーサが含まれます。どちらも正しく定義された XML ファイルを扱い、正当性チェックはおこないません。SAX (Simple API for XML) の実装は、SAX2 の Java による実装と命名規則に準拠します。DOM (Document Object Model) Level 2 の実装は W3C 勧告に準拠し、名前空間をサポートします。

データの保存を XML 形式でおこなう Qt アプリケーションは数多くあります。SAX パーサはデータを先頭から順番に読み込むため、パースがシンプルで巨大なファイルの読み込みに適しています。DOM パーサはファイル全体を読み込み、ツリー構造をメモリ上に展開してトラバースする場合に適しています。

11.3 プロセス間通信

`QProcess` は Qt アプリケーションから他のプログラムを実行し、そのプロセスと通信するためのプラットフォームに依存しない方法を提供します。プロセス間の通信は外部プロセスの標準入力に書き込み、標準出力と標準エラー出力から読み出すという方法でおこなわれます。

`QProcess` は、シグナルを送出することでデータの状態を非同期に通知することができます。Qt アプリケーションはシグナルに接続してプロセスのデータを受け取り、必要であればデータをプロセスに送り返すことができます。

11.4 ネットワーク

Qt は、TCP/IP クライアントおよびサーバを作成するためのプラットフォームに依存しない枠組みが用意されています。

`QSocket` は非同期でバッファリングされた TCP 接続を扱います。`QSocket` は `QIODevice` を継承しているので、ソケットに `QTextStream` と `QDataStream` を使うことができます。

`QSocket` は GUI アプリケーションで利用するのに適した設計になっています。たとえば、リアルタイムで通貨を変換するアプリケーションは次のようになるでしょう。



図 35. リアルタイム通貨変換プログラム

このアプリケーションは、架空の CCP (Currency Conversion Protocol) というプロトコルを使い、サーバから最新のレート情報を受け取ります。ネットワークに関連するコードは次のようにたった数行です。

```
socket = new QSocket( this );
connect( socket, SIGNAL(readyRead()),
this, SLOT(updateTargetAmount()) );
```

ソケットは Converter クラスのコンストラクタで作成されます。ソケット通信は非同期で、ソケットはデータの読み出し準備が整ったとき readyRead() シグナルを送出します。

```
void Converter::convert()
{
    QString command = "CONV " + sourceAmount->text() + " " +
        sourceCurrency->currentText() + " " +
        targetCurrency->currentText() + "\r\n";
    socket->connectToHost( "ccp.banca-monica.nu", 123 );
    socket->writeBlock( command.latin1(), command.length() );
}
```

convert() スロットはユーザが「Convert」ボタンをクリックしたときに呼び出されます。接続を確立し、IP アドレス ccp.banca-monica.nu の 123 番ポート CONV リクエスト ("CONV100 EUR USD" などのようになるでしょう) を送信します。どの処理も非ブロッキングでおこなわれるため、ユーザインターフェイスの応答性が落ちることはありません。

```
void Converter::updateTargetAmount()
{
    if ( socket->canReadLine() ) {
        targetAmount->setText( socket->readLine() );
        socket->close();
    }
}
```

updateTargetAmount() 関数はサーバが CONV リクエストに対する応答を返したときに呼び出されます。updateTargetAmount() は応答を受け取り、表示を更新し、接続を終了します。

TCP サーバは QServerSocket をサブクラス化して作成します。QServerSocket は QSocket と同様非同期で動作します。リスニングソケットをセットアップし、接続を受け付け、クライアントに応答するための仮想関数を呼び出します。

QSocketDevice はシステム固有のソケット API を隠蔽し、プラットフォームに依存しないラップです。QSocket と QServerSocket の基本機能を提供します。QSocketDevice は UDP でも使うことができます。

オンラインリファレンス

<http://doc.trolltech.com/3.2/xml.html>

<http://doc.trolltech.com/3.2/datastreamformat.html>

12 コレクションクラス

コレクションクラスは複数のオブジェクトを格納するためのクラス群です。Qt は STL (Standard Template Library) と互換性のあるコレクションクラスを備えており、コンパイラが STL をサポートするかどうかにかかわらず利用することができます。

アプリケーションでは、複数のオブジェクトのメモリ上に格納する必要が生じることがよくあります。たとえば画像、ウィジェット、カスタムオブジェクトなどです。多くの C++ コンパイラは STL をサポートしています。STL は、データを格納するためによく考えられたデータ構造を持っています。Qt は、STL と同じ構文で利用できるリスト、スタック、キュー、辞書を用意しています。Qt のコレクションクラスは STL 対応のコンパイラでも非対応のコンパイラでも、どちらでも利用することができます。

Qt には豊富で汎用的なコレクションクラス (コンテナ) と、関連するイテレータが用意されており、内部処理や Qt API で非常に多用されています。Qt のコンテナクラスは、「プライベートクラス」と「暗黙的共有」という 2 つの技法を使うことで、速度とメモリ使用効率の点で高度に最適化されています。STL がサポートされる環境では STL コンテナを使うこともできますが、Qt の最適化されたコンテナクラスの恩恵にあずかることができなくなってしまいます。

テンプレートクラスは、実行可能ファイルのサイズを劇的に増加させてしまいます。コンパイラは、それぞれの特化クラスごとに実質的にまったく同じコードを生成しなければならないからです。Qt のテンプレートクラスは、非テンプレートクラスであるプライベートクラスをもとにごく小規模なテンプレートレイヤを実装しているだけなので、生成されるコードは非常に小さくて済むのです。

12.1 値ベースのコレクションクラス

Qt には値ベースのコレクションクラスが 5 つ用意されています。QMap<Key,T>、QValueList<T>、QValueStack<T>、QValueVector<T>、QStringList です。STL のコンテナと非常によく似たインターフェイスを持ち、STL のアルゴリズムと完全に互換性を持っています。Qt には qCopy()、qFind()、qHeapSort() といった STL アルゴリズムと同等のメソッドも用意されています。STL をサポートする環境では STL コンテナと Qt コンテナの間で相互に自動的な変換がおこなわれます。

Qt の値ベースのコレクションクラスでは、暗黙的な共有 (コピーオンライト) という技法が使われています。インスタンスのコピーは同じメモリ上のデータを共有するだけです。データの共有は自動的に処理がおこなわれます。たとえばコピーされたオブジェクトの一方の内容を変更する瞬間にデータの深いコピーがおこなわれるため、もう一方のオブジェクトは何の影響も受けません。オブジェクトがコピーされるときにはポインタのコピーと参照カウントのインクリメントがおこなわれるだけなので、データを実際にコピーするよりもはるかに高速で、メモリ使用量も少なく済みます。

データ共有することが意味を持つあらゆるクラスでデータ共有がおこなわれます。Qt の値ベースのコレクションクラスはもとより、QBitmap、QBrush、QCursor、QFont、QIconSet、QPalette、QPen、QPicture、QPixmap、QRegion、QRegExp、QString などです。こうしたクラスの値コピーは、ポインタ操作や手動での最適化といったリスクを冒すことなく安全に、かつ効率的におこなうことができます。特に QString クラスの暗黙的共有により、文字列の操作を簡単で高速におこなえます。

Qt では低レベルな QMemArray<T>と、その継承クラスである QBitArray、QByteArray、QPointArray というクラスも用意しています。これらのクラスは、従来の基本的なデータ型を非常に効率よく扱うためのクラスです。

12.2 ポインタベースのコレクションクラス

Qt には低レベルで汎用的な、ポインタベースのコレクションクラスも用意されています。QDict<Key,T>、QPtrList<T>、QPtrQueue<T>、QPtrStack<T>、QPtrVector<T>、QCache<T> です。値ではなくポインタを格納するクラスで、特に QWidget や QObject のポインタを格納する際に役立ちます。ポインタベースのコレクションクラスは、コレクションに格納されるオブジェクトを所有することもでき、コレクションクラスが破棄されるとき、コレクションクラスが持っているオブジェクトも同時に自動的に破棄することができます。

オンラインリファレンス

<http://doc.trolltech.com/3.2/qt1.html>

<http://doc.trolltech.com/3.2/collections.html>

<http://doc.trolltech.com/3.2/shclass.html>

13 プラグインと動的ライブラリ

Qt は、動的ライブラリに含まれる関数にアクセスするプラットフォームに依存しない方法を用意します。また、プラグイン機構を備えているため、コーデック、データベースドライバ、画像フォーマットコンバータ、スタイル、カスタムウィジェットなどを開発し、独立したコンポーネントとして配布することも簡単です。

13.1 プラグイン

Qt のコーデック、データベースドライバ、画像フォーマットコンバータ、スタイル、カスタムウィジェットなどをプラグインに変換するには、プラグインの適切なベースクラスをサブクラス化し、単純な関数をいくつか実装し、マクロを追加するだけです。

たとえば、QStyle をサブクラス化し、CopperStyle というクラスを作成するとしましょう。CopperStyle をプラグインとして作成するには、次のようにプラグインクラスをサブクラス化します。

```
class CopperStylePlugin : public QStylePlugin
{
public:
    CopperStylePlugin() { }
    ~CopperStylePlugin() { }

    QStringList keys() const {
        return QStringList() << "CopperStyle";
    }

    QStyle* create( const QString& key ) {
        if ( key == "CopperStyle" )
            return new CopperStyle;
        return 0;
    }
};
Q_EXPORT_PLUGIN( CopperStylePlugin )
```

新しく作成したスタイルを利用するには、次のようにします。

```
QApplication::setStyle( QStyleFactory::create("CopperStyle") );
```

データベースドライバ、コーデック、カスタムウィジェット、画像フォーマットハンドラをプラグインとして作成しておけば、アプリケーションが自動的に検出し、利用できるようになります。

コンパイル済みの動的ライブラリがプラグインとして Qt コンポーネントを開発している企業もすでにあります。



図 36. Klaralvdalens Datakonsult の商用コンポーネントの 1 つ

オンラインリファレンス

<http://doc.trolltech.com/3.2/plugins-howto.html>

13.2 動的ライブラリ

QLibrary クラスを使えば、プラットフォームに依存せず動的ライブラリのロードをおこなうことができます。動的ライブラリを使えば、静的リンクに比べ、はるかに制約の少ない動作が実現できます。ライブラリの動的ロードと使い方のもっとも基本的な方法を以下のサンプルに示します。このサンプルでは mylib というライブラリ (Windows 環境では mylib.dll、Unix 環境では mylib.so) から print_str というシンボルへのポインタを取得します。

```
typedef void (StrFunc)( const char* str );
QLibrary lib( "mylib" );
StrFunc* func = (StrFunc*) lib.resolve( "print_str" );
if ( func )
    func( "Hello world!" );
```

この方法で関数を呼び出すと、型保障がおこなわれず、C リンケージのシンボルしか使えません (C++ 言語の名前変形規則はサポートされないため)。

14 プラットフォーム固有の拡張

Qt は、プラットフォームに共通の機能に加え、プラットフォーム固有の拡張機能を備えており、特定のプラットフォームでの開発を支援する機能を持っています。ActiveQt 機能拡張を使うことで、Qt アプリケーションから ActiveX を使うこと、および Qt アプリケーションを ActiveX サーバとして動作させることが可能となります。Motif 機能拡張を使うことで、Qt と Motif を共存させることが可能になります。

14.1 ActiveQt

ActiveX は Microsoft 社の COM を基盤として作り上げられています。ActiveX は、アプリケーションやライブラリからコンポーネントを使うためのメカニズムです。コンポーネントはコンポーネントサーバから供給されます。アプリケーション自体をコンポーネントサーバとすることもできます。Qt/Windows の ActiveQt モジュールを使えば、アプリケーションを ActiveX サーバに変換したり、他のアプリケーションが提供する ActiveX コントロールを使うことも簡単におこなえます。

ActiveQt は ActiveX をシームレスに Qt に統合します。ActiveX のプロパティ、メソッド、イベントは、それぞれ Qt のプロパティ、スロット、シグナルに対応します。このように直観的なアプローチを採用したことで、Qt 開発者はこれまで慣れ親しんだプログラミング技法を使って ActiveX を利用することができます。また、ActiveX を使うときに避けて通れない、さまざまな自動生成ソースコードを直接扱う必要がなくなります。

Internet Explorer を ActiveX コンポーネントとして登録するサンプルを示します。

```
#define CLSID_InternetExplorer "{8856F961-340A-11D0-A96B-00C04FD705A2}"
QAxWidget *activeX = new QAxWidget( this );
activeX->setControl( CLSID_InternetExplorer );
```

ユーザの操作を追跡したければ、次のようにタイトルバーの変更を検出することもできます。

```
connect( activeX, SIGNAL(TitleChange(const QString&)),
this, SLOT(setTitle(const QString&)) );
```

ActiveQt は、ActiveX と Qt のデータ型を相互に自動的に変換します。

また、dynamicCall() 関数を使って ActiveX コンポーネントを直接制御することもできます。

```
activeX->dynamicCall( "Navigate(const QString&)",
"http://doc.trolltech.com" );
```

また、低レベルな IDispatch インターフェイスもサポートされます。

Qt アプリケーションを ActiveX サーバに変換するのも簡単です。最低限必要なことは、クラスを 1 つエクスポートすることです。qaxfactory.h をインクルードし、QAXFACTORY_DEFAULT マクロを記述します。クラスをコンパイルすれば、プロパティ、スロット、シグナルは ActiveX のプロパティ、メソッド、イベントに変換され、ActiveX クライアントからアクセス可能となります。ActiveQt は QAxFactory::isServer() というメソッドも用意しています。これは、アプリケーションが ActiveX サーバとして動作しているのか、ActiveX コントロールとして動作しているのかを判別するメソッドです。このメソッドを利用して、アプリケーションが実行されている環境に応じて動作を制御することができます。

オンラインリファレンス

<http://doc.trolltech.com/3.2/activeqt.html>

14.2 Motif

大規模な Unix アプリケーションの多くが Motif を使って作成されています。Motif はすでに開発が終了しています。Motif アプリケーション全体を移植する作業は膨大な作業量の開発が必要で、大きなリスクをとまいません。Trolltech 社は、Motif に縛られているユーザに、Qt/Motif 機能拡張という解決策を提供します。

Qt/Motif 機能拡張を使うことで、日常的な保守や開発作業の一環として Motif アプリケーションの細部を 1 つずつ細かく移植していくことができます。このような手法を採れば、移植に必要な人的リソースとリスクを最小に抑えることができます。これは、Qt/Motif モジュールはコードを混在させることができるよう設計されているためです。必要であれば、Motif のイベントループを使いつづけることも、Qt のイベントループに置き換えることもできます。モーダルダイアログ、タイマ、ソケットなどもコード混在環境でまったく問題なく動作します。たとえばあるダイアログを修正しなければならないとしましょう。Qt Designer を使って、簡単に素早く Qt ダイアログに置き換えることもできます。

オンラインリファレンス

<http://doc.trolltech.com/3.2/motif-extension.html>

15 Qt のアーキテクチャ

Qt の機能はすべて、サポートされているプラットフォームの低レベルな API を使って作成されています。このことによって、Qt は柔軟性と効率性を兼ね備えることができます。

Qt は「エミュレーション」型のマルチプラットフォームツールキットです。すべてのウィジェットは Qt 自身が描画します。仮想関数を記述するだけでウィジェットのカスタマイズや拡張がおこなえます。Qt のウィジェットは、サポートされている各プラットフォームのルック&フィールを正確に再現するようにエミュレートをおこないます。詳しくは「9. スタイルとテーマ」を参照してください。この技法によって、既存のスタイルをカスタマイズしてカスタムスタイルを作成することで、アプリケーションに独自のルック&フィールを持たせることができます。

Qt Application Source Code			
Qt API			
Qt/Windows	Qt/X11	Qt/Macintosh	Qt/Embedded
GDI	Xlib	Carbon	
MS-Windows	Unix/Linux	Mac OS X	Embedded Linux

図 37. Qt のアーキテクチャ

Qt は各プラットフォームの低レベルな API を使う点が、従来の「レイヤ」型のマルチプラットフォームツールキットと異なります。従来型のマルチプラットフォームツールキット (Windows の MFC や X11 の Motif) は、いずれか 1 つのプラットフォームのツールキット上に構築されたラッパにすぎません。この方式では、1 つの関数呼び出しだけでも多くのライブラリ呼び出しがおこなわれ、異なる API レイヤの関数呼び出しが連続するため、動作が遅くなります。また、レイヤ型のツールキットは基盤となるツールキットに制約されて柔軟性に欠け、異なるプラットフォームでは微妙に外見が異なることがほとんどで、厄介なバグの元となることが多くあります。

Qt はプロフェッショナルユース向けに作成されており、Microsoft Windows、X11、Mac OS X、組み込み Linux といった各プラットフォームの利点を生かすことができます。1 つのソースコードをターゲットのプラットフォーム上でコンパイルし直すだけで、そのプラットフォーム対応の Qt アプリケーションを作成することができます。Qt はマルチプラットフォームに対応したツールキットですが、プラットフォーム固有のツールキットよりもはるかに学習が簡単で、生産性が高いものであることがお分かりいただけます。Qt の完全なオブジェクト指向アプローチを気に入り、1 つのプラットフォーム向けの開発でも Qt をお使いいただいている顧客も数多くいらっしゃいます。

15.1 Microsoft Windows

Qt/Windows は Win32 API と GDI を使ってイベント取得とプリミティブの描画をおこないます。Qt は MFC を含めどんなツールキットも使いません。特に、柔軟性に欠けるコモンコントロールは使わず、強力でカスタマイズが容易なウィジェットを採用しています (特殊な用途以外では、ファイルダイアログと印刷ダイアログは Windows ネイティブのものを使います)。

Qt を使えば、Windows 95、98、NT4、ME、2000、XP で同じ実行可能ファイルを使うことができます。Qt は実行時に Windows のバージョンをチェックし、そのプラットフォームにもっとも適した機能を利用します。たとえば、文字列の回転は Windows NT4、2000、XP でしかサポートされません。しかし、

Qt は Windows のすべてのバージョンで文字列の回転をおこなうことができます。ネイティブの機能で文字列の回転がサポートされていればそれを利用します。この例でお分かりのように、Qt を使うことで Windows API の細かな違いにわずらわされることがなくなります。

Qt は、Microsoft のアクセシビリティもサポートします。Windows のコモンコントロールとは違い、Qt のウィジェットはベースとなるウィジェットが持つアクセシビリティ情報を失うことなく拡張することができます。つまり、カスタムウィジェットもアクセシビリティ情報を持つことができます。

Qt/Windows では、Microsoft Visual C++ または Borland C++ を使って Qt アプリケーションを作成することができます。

15.2 X11

Qt/X11 は Xlib を使って X サーバと直接通信をおこないます。Qt は Xt (X Toolkit)、Motif、Athena を含めてどんなツールキットも使いません。

Qt アプリケーションは、ウィンドウマネージャやデスクトップ環境に応じて自動的にルック&フィールを変更します。Motif、SGI、CDE、GNOME、KDE といったデスクトップ環境では、ネイティブのルック&フィールで表示がおこなわれます。他の Unix ツールキットでは独自のルック&フィールにユーザを縛りつけてしまうのと対照的です。

Qt は Unicode を完全にサポートします。Qt アプリケーションは Unicode フォントと非 Unicode フォントをサポートします。複数の X フォントを使って多言語テキストをレンダリングすることができます。フォント処理は自律的で、インストールされているフォントと文字セットをすべて検索し、指定されているフォントでは表示できない文字を検出することができます。

また、X エクステンションが利用できる場合には、その機能も利用します。Qt は アンチエイリアスフォントとアルファブレンディングのために RENDER エクステンションを利用します。X インпутメソッドを利用してオンザスポット編集も可能です。マルチスクリーンもサポートし、従来のマルチヘッド機能と Xinerama の両方をサポートします。

Qt は、AIX、BSDI、FreeBSD、HP-UX、Irix、Linux、NetBSD、OpenBSD、Solaris、Tru64、UnixWare の Unix をサポートします。 <http://www.trolltech.com/products/platforms/> で、サポートされているコンパイラと OS の最新情報をご覧ください。

15.3 Mac OS X

バージョン 3.0 から Qt は Carbon API を使って Mac OS X をサポートするようになりました。Qt/Mac を使うことで、Qt アプリケーションの市場がさらに広がります。

Macintosh は、洗練された API を持ち、ビジュアル開発環境は Qt よりも優れていますが、他のプラットフォームで利用することはできません。Qt は、Macintosh でプラットフォームに依存しないレイアウト機能と共通の国際化をもたらします。ファイル処理、ソケットの非同期 I/O、イベントループといった機能、堅牢なデータベースサポートなども Qt の利点です。現代的なオブジェクト指向 API を使い、よく整備されたドキュメントと完全なソースコードを使って Macintosh アプリケーションを開発することができます。

Macintosh の開発者は、自分の気に入ったプラットフォームを使いながら、たとえば Windows など、他の莫大な市場に参入することができます。それも、Windows に依存する特別な機能を別にすれば、ただ再コンパイルするだけです。

Qt/Mac を使うことで、Macintosh 上の開発に標準的な OpenGL、国際化、Qt Designer を使ったビジュアル開発といった多くの利点をもたらすことができます。

15.4 Embedded Linux

Qt/Embedded は独自のウィンドウシステムを持ち、Linux のフレームバッファに直接描画することができます。OpenGL が不要であれば、Qt/Embedded を使うことで、X サーバが不要になり、X11 ベースの組み込み Linux 機器よりもはるかに高速で少ないメモリで動作します。

Qt/Embedded では画像の描画にアルファブレンディングを利用でき、TrueType フォントと Type1 フォントのアンチエイリアスをサポートします。Qtopia と呼ばれる完全な組み込みデバイス向けの開発環境も提供しています。Qtopia は、アプリケーション開発を支援するプログラムランチャ、アプリケーションセット、ライブラリを含みます。手書き文字認識、ピックアップ、仮想キーボードといった柔軟な入力支援機能も備えており、新しい入力方式の開発も容易です。Qtopia はシャープの PDA、Zaurus の標準開発環境にも採用されています。各機能はメモリ量に応じて選択ができるようになっており、Qt/Embedded は 800Kb から 3Mb の範囲でチューニングすることができます。

Qt/Embedded の技術的な詳細については、Qt/Embedded のホワイトペーパーをご覧ください。

16 Qt を使った開発

Qt の開発コミュニティには世界中の企業、開発者が参加しており、毎日活発な議論がおこなわれています。Qt のアーキテクチャが迅速なアプリケーション開発に適していると開発コミュニティに認められたと言ってよいでしょう。こうした開発者は、1 つのプラットフォーム向けでもマルチプラットフォーム向けでも、一貫性があり直観的な Qt の API や、Qt Designer や qmake といった強力なツールの恩恵を受けています。

Qt のユーザコミュニティは活発で情報が豊富です。ユーザたちは qt-interest メーリングリストで議論をおこなっています。メーリングリストの購読やアーカイブの閲覧は <http://lists.trolltech.com/qt-interest/> を参照ください。また、Qt の顧客には、年に 4 回 Qt Quarterly というニュースレターを発送しています。ニュースレターについては <http://doc.trolltech.com/qq/> をご覧ください。

Qt の豊富なドキュメントは <http://doc.trolltech.com> からオンラインでご覧いただくことができます。

Qt の評価版も用意してあります。各プラットフォームごとに 30 日間の評価ができます。 <http://www.trolltech.com> をご覧ください。より詳しい情報をお求めの方は qt-sales@sra.co.jp にメールでお問い合わせください。