

Introdução à Programação C++ com Qt 4

Antonio Marcio A. Menezes

antonio-marcio.menezes@serpro.gov.br

II Fórum de Tecnologia em Software Livre
SERPRO - Regional Porto Alegre

Outubro de 2009

Agenda

1

Parte Teórica

- Por que C++ e Qt?
- O que preciso para começar?

2

Prática - Construção de uma Agenda de Telefones

- Roteiro para a Prática
- Criando o projeto com o Qt Creator
- Trabalhando com Widgets, Layouts, Actions Etc.
- Conectando Signals e Slots
- Acesso a Bancos de Dados
- Traduzindo a Aplicação
- Construindo o instalador da Aplicação

Agenda

- 1 Parte Teórica
 - Por que C++ e Qt?
 - O que preciso para começar?
- 2 Prática - Construção de uma Agenda de Telefones
 - Roteiro para a Prática
 - Criando o projeto com o Qt Creator
 - Trabalhando com Widgets, Layouts, Actions Etc.
 - Conectando Signals e Slots
 - Acesso a Bancos de Dados
 - Traduzindo a Aplicação
 - Construindo o instalador da Aplicação

Agenda

1

Parte Teórica

- Por que C++ e Qt?
- O que preciso para começar?

2

Prática - Construção de uma Agenda de Telefones

- Roteiro para a Prática
- Criando o projeto com o Qt Creator
- Trabalhando com Widgets, Layouts, Actions Etc.
- Conectando Signals e Slots
- Acesso a Bancos de Dados
- Traduzindo a Aplicação
- Construindo o instalador da Aplicação

Qt, o que é isso?

Qt, **muito mais que uma biblioteca**, é um framework para desenvolvimento multi-plataforma de aplicações.

Qt está disponível para as seguintes plataformas:

- Windows.
- Windows CE.
- Linux/X11.
- Linux Embarcado.
- S60 (Symbian - Em Breve).
- MacOS X.

História do Qt

O framework Qt começou a estar disponível publicamente a partir de maio de **1995**. Foi criado por **Haarvard Nord** e **Eirik Chambe-Eng**, fundadores da empresa **Trolltech**. Desde seu início, Qt foi concebido para ser multiplataforma.

Em **março de 1996**, a agência espacial europeia se tornou o segundo cliente do Qt. E em **setembro deste ano**, foi lançada a **versão 1.0** do Qt. Ainda neste ano, foi iniciado o **projeto KDE**, desenvolvido com Qt, por **Matthias Ettrich**. Mais tarde, em **1998**, Matthias foi contratado pela Trolltech.

História do Qt

O framework Qt começou a estar disponível publicamente a partir de maio de **1995**. Foi criado por **Haarvard Nord** e **Eirik Chambe-Eng**, fundadores da empresa **Trolltech**. Desde seu início, Qt foi concebido para ser multiplataforma.

Em **março de 1996**, a agência espacial europeia se tornou o segundo cliente do Qt. E em **setembro deste ano**, foi lançada a **versão 1.0** do Qt. Ainda neste ano, foi iniciado o **projeto KDE**, desenvolvido com Qt, por **Matthias Ettrich**. Mais tarde, em **1998**, Matthias foi contratado pela Trolltech.

História do Qt

Em **1999**, a **versão 2.0** do Qt foi liberada. Uma versão do Qt para linux embarcado, por sua vez, foi lançada em **2000**.

Em **2001**, seguiu-se o lançamento da **versão 3.0** do Qt, agora disponível para Windows, MacOS X, Unix e Linux (desktop e embarcado).

Em meados de **2005**, a **versão Qt 4.0** tornou-se disponível, contando com 500 classes e mais de 9000 funções.

História do Qt

Em **1999**, a **versão 2.0** do Qt foi liberada. Uma versão do Qt para linux embarcado, por sua vez, foi lançada em **2000**.

Em **2001**, seguiu-se o lançamento da **versão 3.0** do Qt, agora disponível para Windows, MacOS X, Unix e Linux (desktop e embarcado).

Em meados de **2005**, a **versão Qt 4.0** tornou-se disponível, contando com 500 classes e mais de 9000 funções.

História do Qt

Em 1999, a versão 2.0 do Qt foi liberada. Uma versão do Qt para linux embarcado, por sua vez, foi lançada em 2000.

Em 2001, seguiu-se o lançamento da versão 3.0 do Qt, agora disponível para Windows, MacOS X, Unix e Linux (desktop e embarcado).

Em meados de 2005, a versão Qt 4.0 tornou-se disponível, contando com 500 classes e mais de 9000 funções.

História do Qt

Em **2008**, a empresa Trolltech foi adquirida pela **Nokia**. Trolltech agora é chamada de **Qt Software**. Pouco tempo depois, o Qt passou a ser lançado sob as licenças GPL, LGPL e comercial.

Recentemente, além de ser fornecido o framework Qt, a empresa Qt Software passou a fornecer uma IDE amigável para desenvolvimento de aplicações: o **Qt Creator**.

História do Qt

Em **2008**, a empresa Trolltech foi adquirida pela **Nokia**. Trolltech agora é chamada de **Qt Software**. Pouco tempo depois, o Qt passou a ser lançado sob as licenças GPL, LGPL e comercial.

Recentemente, além de ser fornecido o framework Qt, a empresa Qt Software passou a fornecer uma IDE amigável para desenvolvimento de aplicações: o **Qt Creator**.

Por que C++ e Qt?

O que eu ganho em programar com C++ e Qt?

- Desenvolvimento **Multiplataforma**.
- Programação C++ mais amigável **com Qt**.
- Implemente uma vez. **Compile em qualquer lugar**.
- Aplicações KDE **são feitas com Qt**.
- Criação de interfaces gráficas **elegantes e amigáveis**.
- Utilização de uma API **rica e útil**.
- Licenças **Comercial, LGPL e GPL**.

Quem usa Qt?

Em Aplicações Desktop

- **Phoenix**, um sistema para controle de agendamento e monitoração de vôos, desenvolvido pela DFS (empresa de controle aéreo da Alemanha).
- **VLC**, reprodutor de mídia.
- **Google Earth**, para visualização de imagens de satélite e outras informações georreferenciadas.
- **Skype**, ferramenta de comunicação de voz e dados via internet.
- **PSI**, programa para envio de mensagens instantâneas.

Quem usa Qt?

Em Sistemas Embarcados

- Porta-retrato digital SPF-105V da **Samsung**.
- Vários modelos de celulares da **Motorola**.
- Vídeo-fone VP 5500 da **Philips**.
- **Sony mylo**, um comunicador pessoal.
- Telefone Wi-Fi KX-WP1050 para **Skype** da **Panasonic**.
- Smartphones 3G da **ZTE**.
- GPS Naviflash 1020 da **Bury**.
- Reprodutor de mídia portátil da **digitalCube**.

Agenda

1

Parte Teórica

- Por que C++ e Qt?
- O que preciso para começar?

2

Prática - Construção de uma Agenda de Telefones

- Roteiro para a Prática
- Criando o projeto com o Qt Creator
- Trabalhando com Widgets, Layouts, Actions Etc.
- Conectando Signals e Slots
- Acesso a Bancos de Dados
- Traduzindo a Aplicação
- Construindo o instalador da Aplicação

O que preciso para começar?

Conhecer C ou C++

Apesar de ser importante um conhecimento mais avançado de C++, para quem programa em Java ou outras linguagens orientadas a objetos é possível apenas estudar as diferenças sintáticas para já iniciar o aprendizado.

Estudar através de exemplos

A instalação do Qt já vem com muitos exemplos de aplicações úteis. Estes exemplos são fornecidos com seus códigos.

Instalar o **Qt SDK** e consultar outras referências (apresentadas ao final deste curso).

Instalação

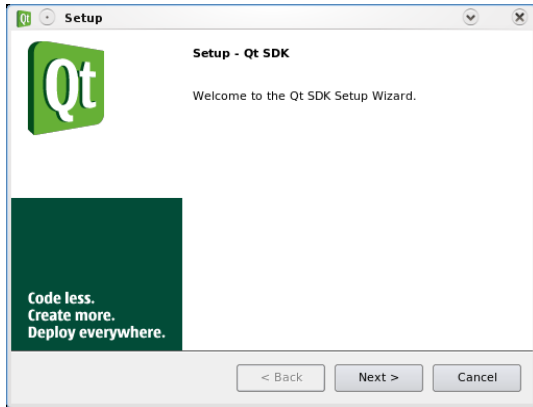
Na página <http://qt.nokia.com/downloads> você pode fazer o download da versão LGPL/Free.

Clique em “Download Qt SDK for Linux/X11 32-bit (275 Mb)”.

Com isto, você irá obter o arquivo de instalação (certifique-se de conceder permissão de execução para o mesmo):
[qt-sdk-linux-x86-opensource-2009.03.1.bin](#).

É necessário também verificar se estão instalados os seguintes pacotes: [libfreetype6-dev](#) e [libgtk2.0-dev](#).

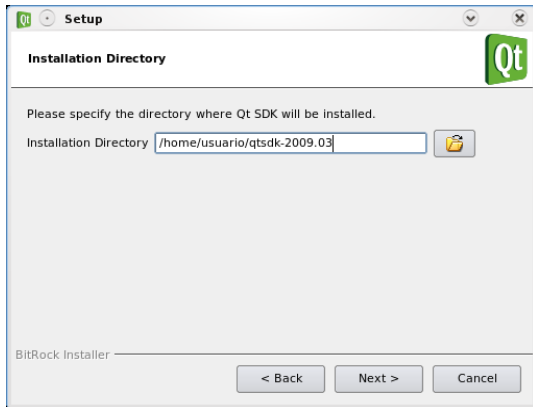
Instalação - Tela 1



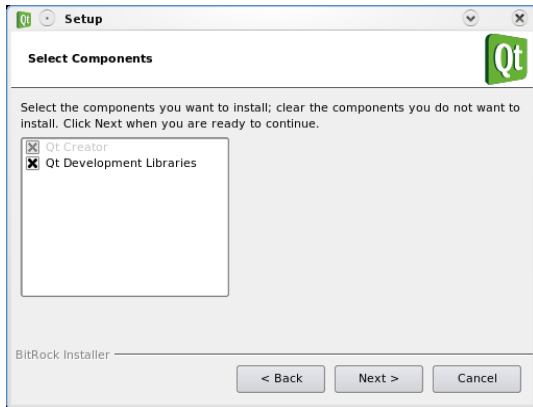
Instalação - Tela 2



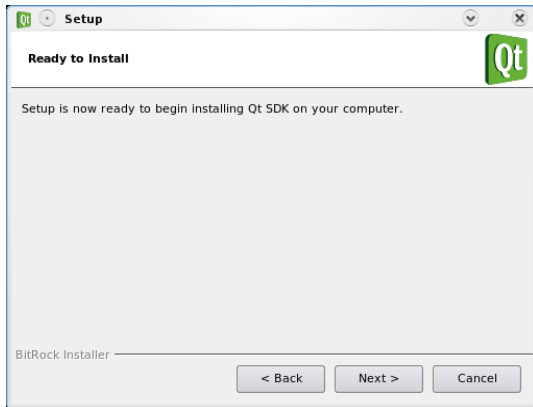
Instalação - Tela 3



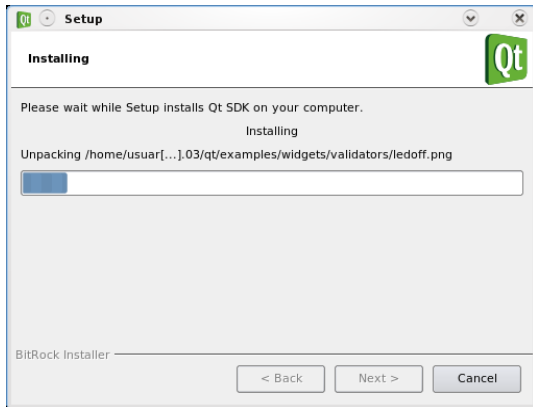
Instalação - Tela 4



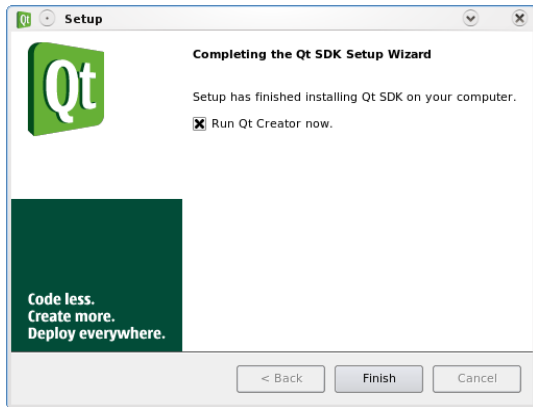
Instalação - Tela 5



Instalação - Tela 6



Instalação - Tela 7



Instalação - Tela 8



Instalação - Resumo

Ao final, no diretório de instalação do Qt SDK, serão encontrados os seguintes subdiretórios:

- **bin** é o diretório onde se encontra o executável do IDE Qt Creator.
- **lib** armazena bibliotecas necessárias a execução do Qt Creator.
- **qt** é onde se encontra o framework Qt e suas ferramentas. Dentro deste, há outro diretório **bin** que contém os executáveis do framework.
- **share** é apenas um diretório de recursos (doc, imagens etc) utilizados pelo Qt Creator.

Fundamentos

Um programa em C++ consiste de um ou mais **unidades de compilação**. Cada unidade de compilação é um arquivo texto de código fonte, tipicamente, com uma extensão **.cpp** (ou **.cc** ou **.cxx**).

Para uma unidade de compilação, o compilador irá gerar um **arquivo objeto** com a extensão **.obj** (no Windows) ou **.o** (no Unix ou MACOS X). O arquivo objeto é um **arquivo binário** com o código de máquina **específico da arquitetura** da máquina onde foi gerado.

Fundamentos

Um programa em C++ consiste de um ou mais **unidades de compilação**. Cada unidade de compilação é um arquivo texto de código fonte, tipicamente, com uma extensão **.cpp** (ou **.cc** ou **.cxx**).

Para uma unidade de compilação, o compilador irá gerar um **arquivo objeto** com a extensão **.obj** (no Windows) ou **.o** (no Unix ou MACOS X). O arquivo objeto é um **arquivo binário** com o código de máquina **específico da arquitetura** da máquina onde foi gerado.

Fundamentos

Um programa em C++ passa por 6 passos

- **Edição** (com um programa editor de texto, por exemplo).
- **Pré-processamento** (através de diretivas são realizadas inclusões e substituições de textos).
- **Compilação** (geração do código objeto).
- **Ligação** ("linking" com o código das rotinas referenciadas em bibliotecas).
- **Carga** (carga do programa e bibliotecas na memória).
- **Execução** (sob controle da CPU, uma instrução por vez).

Fundamentos

Um programa em C++ passa por 6 passos

- **Edição** (com um programa editor de texto, por exemplo).
- **Pré-processamento** (através de diretivas são realizadas inclusões e substituições de textos).
- **Compilação** (geração do código objeto).
- **Ligação** ("linking" com o código das rotinas referenciadas em bibliotecas).
- **Carga** (carga do programa e bibliotecas na memória).
- **Execução** (sob controle da CPU, uma instrução por vez).

Fundamentos

Um programa em C++ passa por 6 passos

- **Edição** (com um programa editor de texto, por exemplo).
- **Pré-processamento** (através de diretivas são realizadas inclusões e substituições de textos).
- **Compilação** (geração do código objeto).
- **Ligação** ("linking" com o código das rotinas referenciadas em bibliotecas).
- **Carga** (carga do programa e bibliotecas na memória).
- **Execução** (sob controle da CPU, uma instrução por vez).

Fundamentos

Um programa em C++ passa por 6 passos

- **Edição** (com um programa editor de texto, por exemplo).
- **Pré-processamento** (através de diretivas são realizadas inclusões e substituições de textos).
- **Compilação** (geração do código objeto).
- **Ligação** (“linking” com o código das rotinas referenciadas em bibliotecas).
- **Carga** (carga do programa e bibliotecas na memória).
- **Execução** (sob controle da CPU, uma instrução por vez).

Fundamentos

Um programa em C++ passa por 6 passos

- **Edição** (com um programa editor de texto, por exemplo).
- **Pré-processamento** (através de diretivas são realizadas inclusões e substituições de textos).
- **Compilação** (geração do código objeto).
- **Ligação** (“linking” com o código das rotinas referenciadas em bibliotecas).
- **Carga** (carga do programa e bibliotecas na memória).
- **Execução** (sob controle da CPU, uma instrução por vez).

Fundamentos

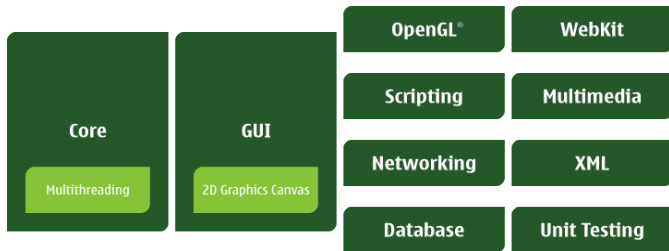
Um programa em C++ passa por 6 passos

- **Edição** (com um programa editor de texto, por exemplo).
- **Pré-processamento** (através de diretivas são realizadas inclusões e substituições de textos).
- **Compilação** (geração do código objeto).
- **Ligação** (“linking” com o código das rotinas referenciadas em bibliotecas).
- **Carga** (carga do programa e bibliotecas na memória).
- **Execução** (sob controle da CPU, uma instrução por vez).

Módulos do Qt

Os módulos que compõem, atualmente, o framework Qt são:

QtCore, QtGui, QtNetwork, QtOpenGL, QtScript, QtSql, QSvg, QtWebKit, QtXml, QtXmlPatterns, Phonon, Qt3Support, QTest, QtDBus.



Ferramentas do Qt

O Qt também vem acompanhado de um conjunto de ferramentas que auxiliam o processo de desenvolvimento.

- **Qt Designer** para o projeto de telas. Permite testar o projeto de tela sem programar qualquer linha de código.
- **Qt Linguist** para editar arquivos de traduções, tornando uma aplicação com suporte a diferentes línguas.
- **Qt Assistant** é um guia de consulta rápida e sensível a contexto para a API do Qt.
- **Qt Creator** é o IDE para a criação de projetos de aplicações.

Ferramentas do Qt

O Qt também vem acompanhado de um conjunto de ferramentas que auxiliam o processo de desenvolvimento.

- **Qt Designer** para o projeto de telas. Permite testar o projeto de tela sem programar qualquer linha de código.
- **Qt Linguist** para editar arquivos de traduções, tornando uma aplicação com suporte a diferentes línguas.
- **Qt Assistant** é um guia de consulta rápida e sensível a contexto para a API do Qt.
- **Qt Creator** é o IDE para a criação de projetos de aplicações.

Ferramentas do Qt

O Qt também vem acompanhado de um conjunto de ferramentas que auxiliam o processo de desenvolvimento.

- **Qt Designer** para o projeto de telas. Permite testar o projeto de tela sem programar qualquer linha de código.
- **Qt Linguist** para editar arquivos de traduções, tornando uma aplicação com suporte a diferentes línguas.
- **Qt Assistant** é um guia de consulta rápida e sensível a contexto para a API do Qt.
- **Qt Creator** é o IDE para a criação de projetos de aplicações.

Ferramentas do Qt

O Qt também vem acompanhado de um conjunto de ferramentas que auxiliam o processo de desenvolvimento.

- **Qt Designer** para o projeto de telas. Permite testar o projeto de tela sem programar qualquer linha de código.
- **Qt Linguist** para editar arquivos de traduções, tornando uma aplicação com suporte a diferentes línguas.
- **Qt Assistant** é um guia de consulta rápida e sensível a contexto para a API do Qt.
- **Qt Creator** é o IDE para a criação de projetos de aplicações.

Minha primeira aplicação com Qt

hello.cpp

```
#include <QApplication>
#include <QLabel>
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QLabel *label = new QLabel(QString::fromUtf8("Olá Qt!"));
    label->show();
    return app.exec();
}
```

Minha primeira aplicação com Qt

Após criar um diretório **hello**, será incluído deste diretório o arquivo **.cpp** da minha primeira aplicação: **hello.cpp**

Ao digitar **qmake -project**, será criado o arquivo de projeto **hello.pro**

qmake hello.pro irá gerar o arquivo **Makefile**, que orientará a compilação do projeto.

Ao digitar **make**, será compilado e gerado o executável da aplicação.

Minha primeira aplicação com Qt

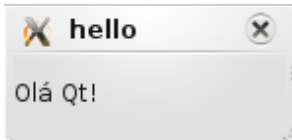


Figura: No Linux

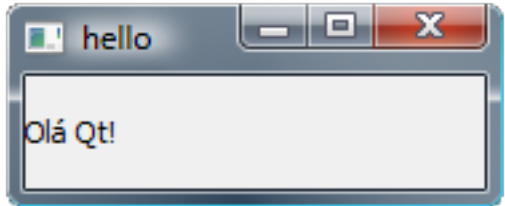


Figura: No Windows

Agenda

1 Parte Teórica

- Por que C++ e Qt?
- O que preciso para começar?

2 Prática - Construção de uma Agenda de Telefones

- **Roteiro para a Prática**
- Criando o projeto com o Qt Creator
- Trabalhando com Widgets, Layouts, Actions Etc.
- Conectando Signals e Slots
- Acesso a Bancos de Dados
- Traduzindo a Aplicação
- Construindo o instalador da Aplicação

Roteiro para a Prática

Neste treinamento, serão exercitadas algumas tarefas comuns no desenvolvimento de um sistema informatizado. Para agilizar o treinamento, serão propostas **versões incrementais** da aplicação exemplo.

Estas versões incrementais poderão ser enxergadas como **pontos de controle**. Quando um aluno concluir a **principal atividade** de um ponto de controle, ele poderá adotar uma versão mais completa da aplicação e seguir para as próximas tarefas.

Roteiro para a Prática

Versões incrementais da aplicação

- **agenda-telefonica1.zip** contém o projeto inicial, sem projeto de tela e apenas um arquivo principal de execução da aplicação.
- **agenda-telefonica2.zip** contém o layout inicial da tela principal da aplicação.
- **agenda-telefonica3.zip** contém a barra de ferramentas da tela principal da aplicação com as ações básicas e ícones.
- **agenda-telefonica4.zip** contém o recurso de ativar a aplicação na área de notificação.

Roteiro para a Prática

Versões incrementais da aplicação

- **agenda-telefonica5.zip** inclui código para evitar que mais de uma agenda seja carregada na memória ao mesmo tempo. São criadas as primeiras conexões entre signals e slots.
- **agenda-telefonica6.zip** contém o primeiro código para acesso ao banco de dados e apresentação dos dados na janela principal.
- **agenda-telefonica7.zip** inclui o projeto das telas de cadastro de departamentos e de telefones, e o armazenamento em banco de dados.

Roteiro para a Prática

Versões incrementais da aplicação

- **agenda-telefonica8.zip** contém a aplicação com suas telas traduzidas para o inglês e também o arquivo de tradução da aplicação para o português.
- **agenda-telefonica9.zip** contém os arquivos necessários da agenda para o instalador da aplicação.
- **agenda-telefonica10.zip** inclui o projeto do instalador com a ferramenta GPL **installJammer**.

Requisitos da Aplicação

Requisitos I

- Cada **peessoa** cadastrada na agenda deverá ser associada a um **departamento** na qual trabalha.
- Todo departamento será identificado por um **código**, uma **sigla** e um **nome**. Siglas e nomes dos departamentos serão alfanuméricos.
- O cadastro de cada pessoa e seu telefone de contato poderá também ser acrescido com a informação de um número de **ramal**.

Requisitos da Aplicação

Requisitos II

- Os dados dos cadastros serão armazenados **localmente**, na máquina onde se executa a agenda.
- A janela principal da agenda deverá fornecer uma **busca simplificada** dos telefones pelos critérios: nome (ou parte do nome) da pessoa e o nome do departamento.
- **Inicialmente**, a janela principal mostrará a lista de todos os telefones, ordenados por nome da pessoa.
- Deverá ser exibido na janela principal também um **contador do número de itens** exibidos nesta lista.

Requisitos da Aplicação

Requisitos III

- Deverá existir uma janela de **busca avançada** onde poderão ser identificadas as pessoas e seus telefones por parte do telefone/ramal ou parte do nome do departamento.
- Quando a janela da agenda for **fechada**, ela deverá ficar ainda ativa, com um ícone na **área de notificação** (**System TrayBar**). Nesta situação, o usuário poderá clicar sobre este ícone e solicitar que a janela seja novamente exibida.

Requisitos da Aplicação

Requisitos IV

- O cadastro dos departamentos será feito em uma janela, **separadamente**. Em outra janela, será feito o cadastro das pessoas e seus telefones.
- Não poderá ser cadastrado mais de um número de telefone por pessoa.
- A empresa que utilizará a agenda de telefones e ramais deseja executar este programa em máquinas com **Linux** ou **Windows**.

Esboço da Janela Principal da Aplicação

AGENDA DE TELEFONES E RAMAIS

Cadastros Pesquisas

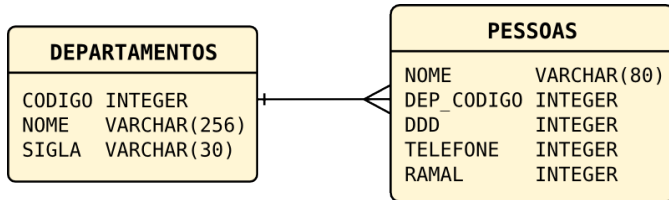
D T P X

Nome Departamento ▼ **Buscar** **Limpar**

NOME	DEPARTAMENTO	TELEFONE	RAMAL

9999 TELEFONE(S) ENCONTRADO(S)

Diagrama ER da Aplicação



Para fins de simplicidade, será utilizado o banco **SQLite**.

Script de criação do banco de dados

```
create table DEPARTAMENTOS (  
    CODIGO INTEGER NOT NULL  
        PRIMARY KEY AUTOINCREMENT,  
    NOME VARCHAR(256) NOT NULL UNIQUE,  
    SIGLA VARCHAR(30) NULL  
);
```

Script de criação do banco de dados

```
create table PESSOAS (  
    NOME VARCHAR(80) NOT NULL PRIMARY KEY,  
    DEP_CODIGO INTEGER NOT NULL  
        REFERENCES DEPARTAMENTOS(CODIGO)  
        ON DELETE RESTRICT ON UPDATE RESTRICT  
        ON INSERT RESTRICT,  
    DDD INTEGER,  
    TELEFONE INTEGER NOT NULL,  
    RAMAL INTEGER  
);
```


Agenda

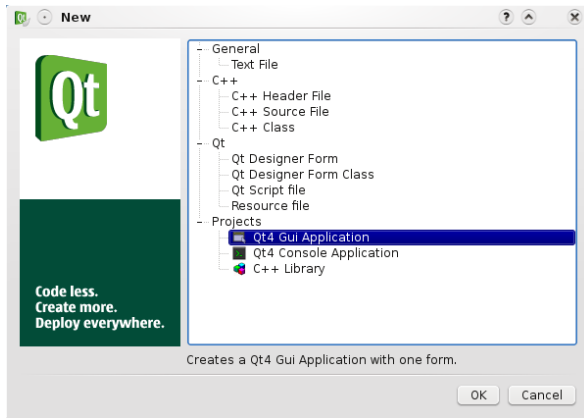
- 1 Parte Teórica
 - Por que C++ e Qt?
 - O que preciso para começar?
- 2 Prática - Construção de uma Agenda de Telefones
 - Roteiro para a Prática
 - **Criando o projeto com o Qt Creator**
 - Trabalhando com Widgets, Layouts, Actions Etc.
 - Conectando Signals e Slots
 - Acesso a Bancos de Dados
 - Traduzindo a Aplicação
 - Construindo o instalador da Aplicação

Exercício de Criação do Projeto

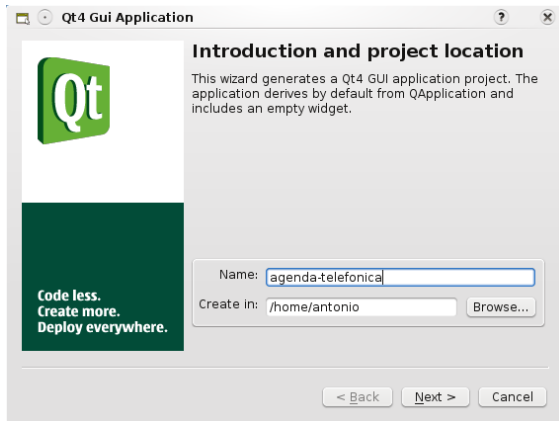
Exercício 01

- **Objetivo:** Aprender como criar um projeto de aplicação GUI (janela do tipo QMainWindow) com o Qt Creator.
- **Tempo Estimado:** 5 a 8 min.
- **Resultado Esperado:** Criar todos os arquivos, como os que foram criados na versão agenda-telefonica1.zip.

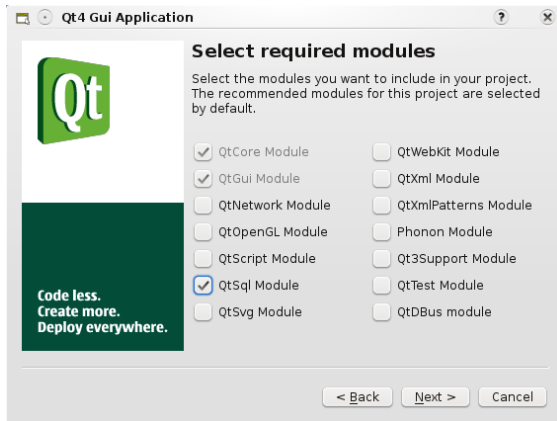
Definindo o tipo de projeto



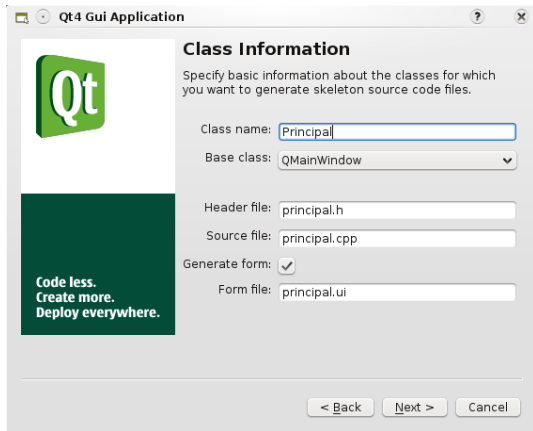
Definindo o nome do projeto



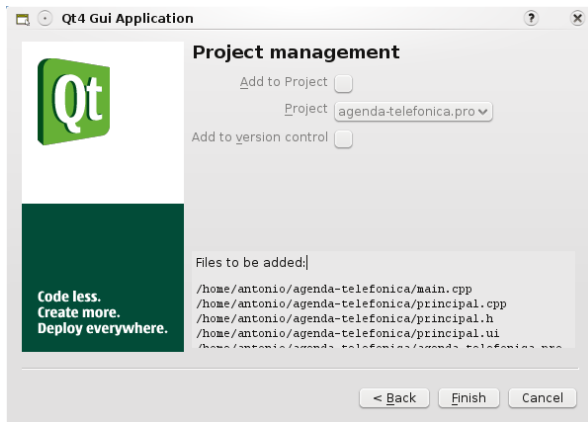
Módulos do Qt que farão parte do projeto



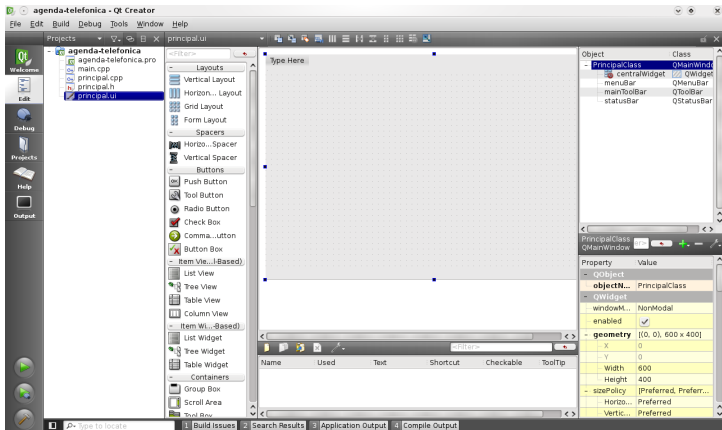
Definindo uma nova classe do tipo QMainWindow



Resumo da criação do projeto



Tela do Qt Creator com o novo projeto



Agenda

- 1 Parte Teórica
 - Por que C++ e Qt?
 - O que preciso para começar?
- 2 Prática - Construção de uma Agenda de Telefones
 - Roteiro para a Prática
 - Criando o projeto com o Qt Creator
 - **Trabalhando com Widgets, Layouts, Actions Etc.**
 - Conectando Signals e Slots
 - Acesso a Bancos de Dados
 - Traduzindo a Aplicação
 - Construindo o instalador da Aplicação

Widgets

Widgets (**Window Gadgets**) são componentes que possuem uma representação gráfica no projeto de telas. Com o **Qt Designer**, o desenvolvedor pode criar interfaces gráficas de usuário apenas arrastando e soltando vários widgets sobre a janela que está sendo projetada.

É possível também testar o funcionamento “visual” da janela sob temas gráficos diferentes (Plastique, GTK+, Motif, CleanLooks etc).

Exercício com Widgets

Exercício 02

- **Objetivo:** Aprender a criar projetos de telas com componentes visuais.
- **Tempo Estimado:** 2 a 5 min.
- **Resultado Esperado:** Adicionar todos os widgets necessários para o projeto da janela principal da Agenda.

Exercício com Widgets - Esboço da Tela

AGENDA DE TELEFONES E RAMAIS

Cadastros Pesquisas

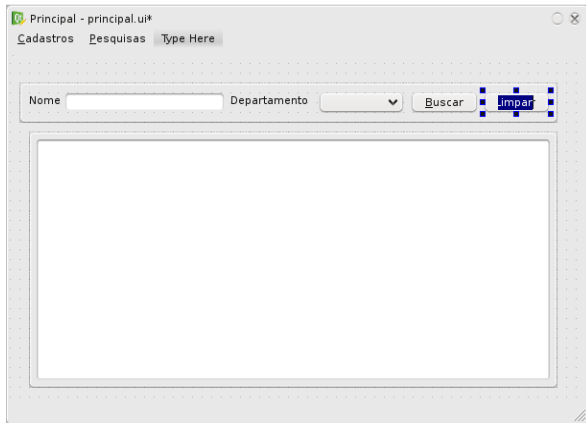
D T P X

Nome Departamento Buscar Limpar

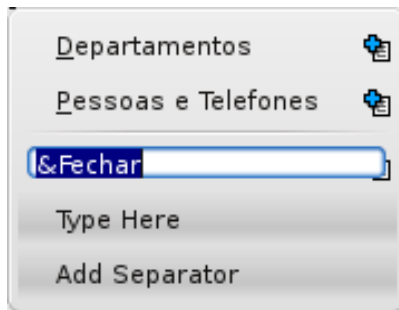
NOME	DEPARTAMENTO	TELEFONE	RAMAL

9999 TELEFONE(S) ENCONTRADO(S)

Exercício com Widgets - Atribuindo textos



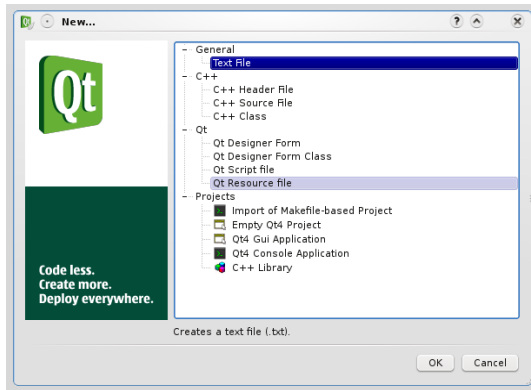
Exercício com Widgets - Atribuindo textos ao Menu



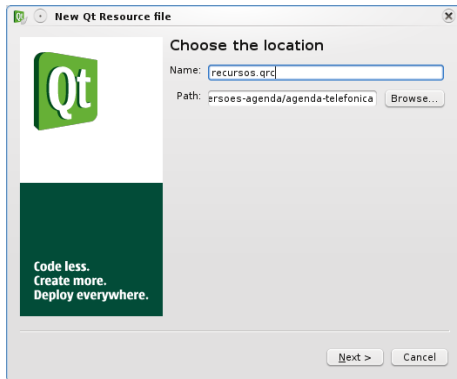
Atribuindo textos ao Menu (QMenu)

Ao adicionar itens ao menu principal, automaticamente serão criadas **Actions** que poderão ser ligadas a métodos da janela principal. Também é possível definir ícones para as Actions, mas para isso é necessário criar um arquivo de recursos.

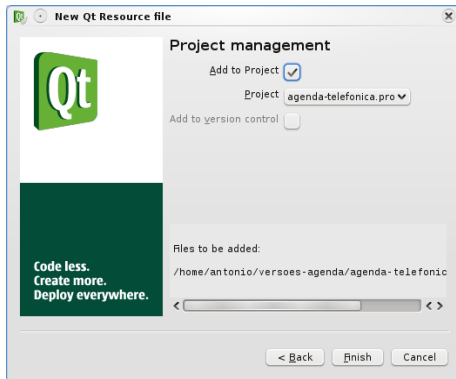
Criando um arquivo de recursos



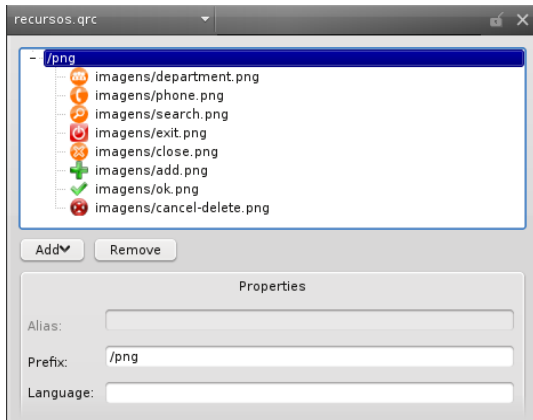
Criando um arquivo de recursos



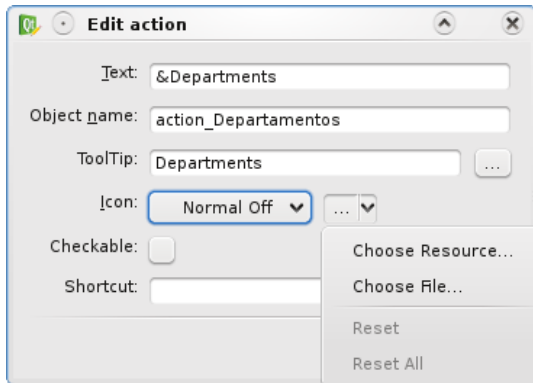
Criando um arquivo de recursos



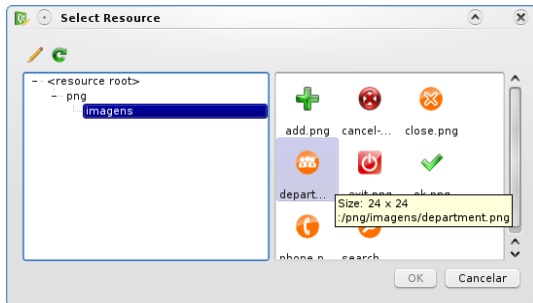
Criando um arquivo de recursos



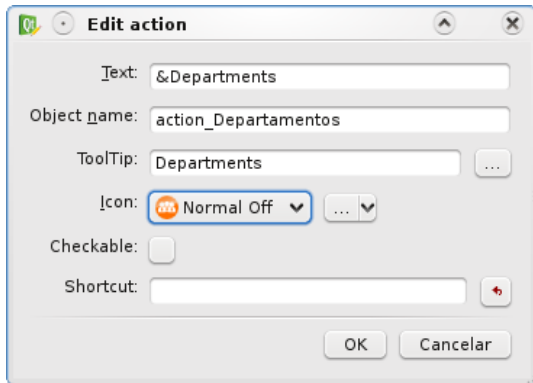
Atribuindo um ícone a uma Action



Atribuindo um ícone a uma Action



Atribuindo um ícone a uma Action



Actions e Barra de Ferramentas

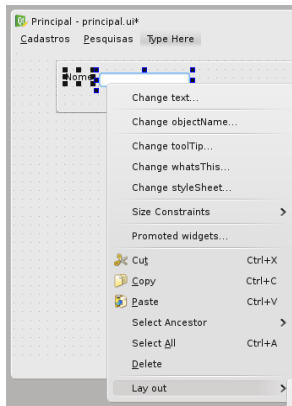
A partir do **painel de actions**, é possível arrastar e soltar actions sobre a **barra de ferramentas** da janela principal. Com isto, serão automaticamente criados **botões** na barra de ferramentas que acionarão as actions ao serem clicados. Os **ícones** das actions também aparecerão nos botões da barra de ferramentas.

Exercício com Layouts

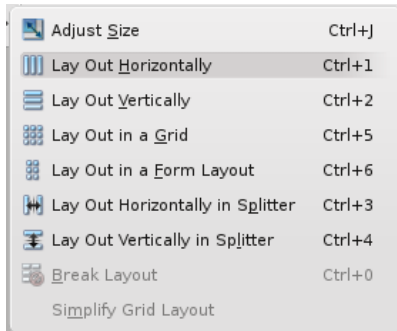
Exercício 03

- **Objetivo:** Aprender a definir os layouts de telas com o Qt Creator / Qt Designer.
- **Tempo Estimado:** 8 a 12 min.
- **Resultado Esperado:** Layout básico de widgets na janela principal.

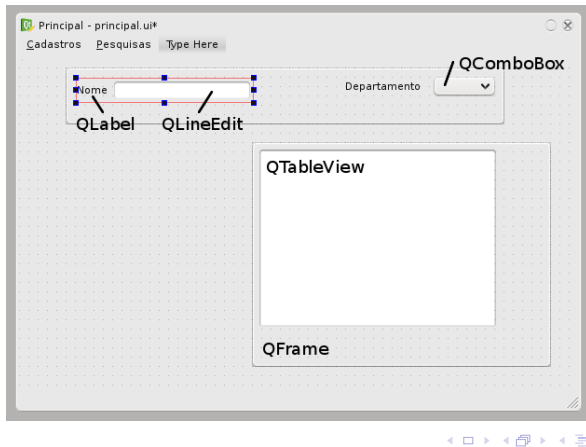
Definindo o primeiro layout



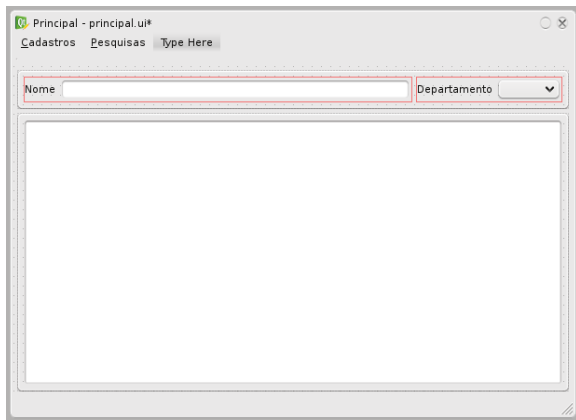
Definindo o primeiro layout



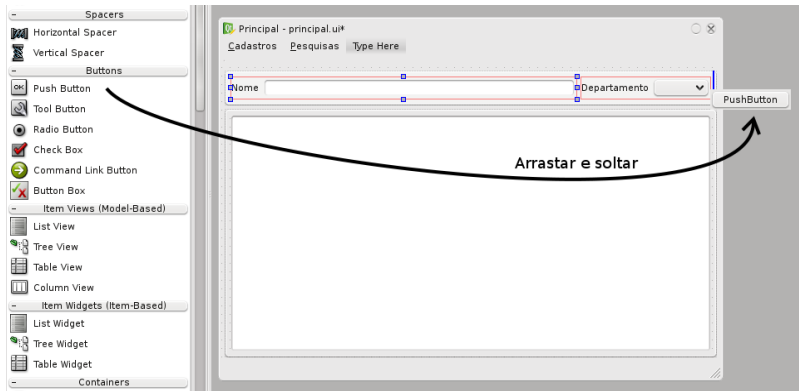
Definindo o primeiro layout



Definindo o primeiro layout



Adicionando um botão a um layout já existente



Agenda

1 Parte Teórica

- Por que C++ e Qt?
- O que preciso para começar?

2 Prática - Construção de uma Agenda de Telefones

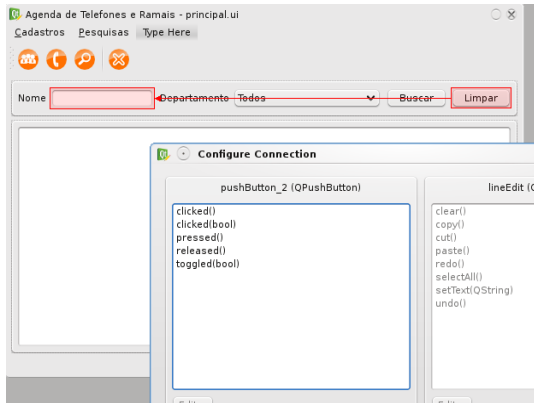
- Roteiro para a Prática
- Criando o projeto com o Qt Creator
- Trabalhando com Widgets, Layouts, Actions Etc.
- **Conectando Signals e Slots**
- Acesso a Bancos de Dados
- Traduzindo a Aplicação
- Construindo o instalador da Aplicação

O que são signals e slots?

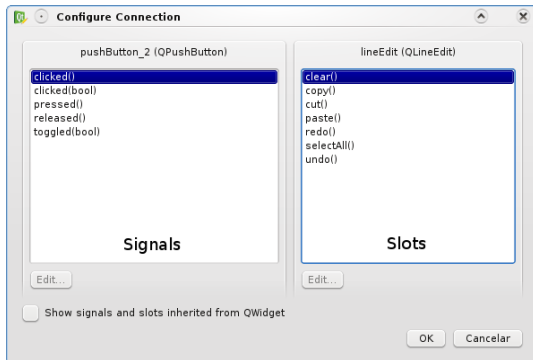
No Qt, um **signal** (sinal) é emitido quando um evento ocorre. Existem signals pré-definidos, mas também é possível de definir seus próprios signals. Um signal pode ser ligado a um método. Desta forma, quando o signal for emitido, este método será executado. Nesta ligação entre signal e método, o método é o **slot**.



Conectando um signal a um slot da janela principal



Definindo signal e slot apropriados



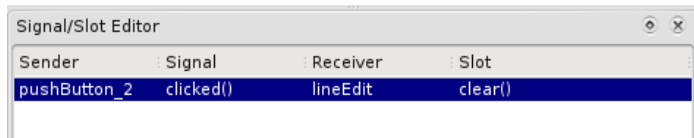
Exercício com Signals

Exercício 04

- **Objetivo:** Aprender a conectar signals e slots dos widgets com Qt Designer.
- **Tempo Estimado:** 2 a 5 min.
- **Resultado Esperado:** O click sobre um **QPushButton** executará o método clean de um **QLineEdit**. Testar este comportamento com o **Form Preview** no Qt Designer.

Signals e Slots no Qt Designer

Num painel (ou janela) do Qt Designer, se pode consultar as conexões já definidas entre sinais e slots.



Conectando signal e slot, programaticamente

Exemplo conectando Signal e Slot

```
void conectarSignals()
{
    connect(
        ui->pushButton_2, SIGNAL(clicked()),
        lineEdit, SLOT(clear())
    );
}
```

Mais sobre signals

Um signal também podem ser conectado a outro signal.

O mecanismo de conexão entre signals e slots também pode resolver o problema de **referência cíclica**. Por exemplo, a **janela principal** pode criar uma **outra janela** e esperar que interações nesta segunda janela determinem mudanças na janela principal.

Este comportamento irá aparecer na aplicação exemplo aqui tratada. Quando forem criados, modificados ou excluídos **departamentos**, a **ComboBox de Departamento** da janela principal deverá ser atualizada.

Mais sobre signals

Trecho do método mostrarCadastroDepartamentos()

```
void Principal::mostrarCadastroDepartamentos()
{
    Departamentos *dep = new Departamentos();
    connect(
        dep, SIGNAL(departamentosAtualizados()),
        this, SLOT(atualizarComboDepartamentos())
    );
}
```

Agenda

1 Parte Teórica

- Por que C++ e Qt?
- O que preciso para começar?

2 Prática - Construção de uma Agenda de Telefones

- Roteiro para a Prática
- Criando o projeto com o Qt Creator
- Trabalhando com Widgets, Layouts, Actions Etc.
- Conectando Signals e Slots
- **Acesso a Bancos de Dados**
- Traduzindo a Aplicação
- Construindo o instalador da Aplicação

Módulo QSql

O módulo QSql é o que fornece as classes e rotinas necessárias para estabelecer conexões a bancos de dados e execução de operações, em diferentes SGBDs: **IBM DB2, Interbase/Firebird, MySQL, Oracle, PostgreSQL, SQLite, Sybase.**

É possível acessar outros bancos com o driver **ODBC**, além de também existirem plugins para acesso a outras bases de dados (xBase, por exemplo).

Criando uma conexão a um banco de dados

```
bool criarConexao() {  
    QSqlDatabase bd = QSqlDatabase::addDatabase("QPSQL");  
    bd.setHostName("sgbd.empresa.com.br");  
    bd.setDatabaseName("BdFuncionarios");  
    bd.setUserName("usuario");  
    bd.setPassword("senha");  
    if (! bd.open() ) {  
        QMessageBox::critical(0, tr("DB Error"), bd.lastError().text());  
        return false; }  
    return true; }
```

Realizando uma consulta no banco

```
QStringList listaTelefones;  
QStringQuery consulta;  
consulta.exec("SELECT TELEFONE FROM PESSOAS");  
while (consulta.next()) {  
    QString telefone = consulta.value(0).toString();  
    listaTelefones.append(telefone);  
}
```

Exercício com Banco de Dados

Exercício 05

- **Objetivo:** Aprender a executar uma conexão e uma consulta num banco de dados.
- **Tempo Estimado:** 10 a 20 min.
- **Resultado Esperado:** Implementar o preenchimento da QComboBox de Departamentos na janela Principal. Os dados desta ComboBox serão obtidos através de consulta que recupera os nomes dos departamentos existentes no banco de dados.

Exercício com Banco de Dados

```
QStringList Principal::getDepartamentos() {  
    QStringList lista;  
    lista.append("Todos");  
    QSqlQuery query( "select distinct NOME from DEPARTAMENTOS  
                      order by NOME", bancoDeDados);  
    while (query.next()) {  
        lista.append(query.value(0).toString());  
    }  
    return lista;  
}
```

Exercício com Banco de Dados

```
void Principal::atualizarComboDepartamentos() {  
    ui->comboDepartamento->clear();  
    ui->comboDepartamento->addItems(getDepartamentos());  
}
```

O método `addItems` de `QComboBox` recebe como parâmetro um `QStringList`.

Atualizando dados em um banco

```
QSqlQuery oper;  
oper.prepare(  
    "insert into DEPART(NOME, SIGLA) values (:NM, :SG)"  
);  
oper.bindValue(":NM", "Desenvolvimento e TI");  
oper.bindValue(":SG", "DTI");  
oper.exec();  
QSqlQuery oper2(  
    "update DEPART set SIGLA='TI' where SIGLA='DTI' ");
```

Classes de Modelo Sql

Para quem prefere uma interação mais distante da sintaxe SQL, o Qt fornece algumas classes específicas.

Classes de Modelo Sql

- **QSqlQueryModel**: Um modelo de dados read-only baseado em uma consulta SQL.
- **QSqlTableModel**: Um modelo read-write que trabalha somente sobre uma única tabela.
- **QSqlRelationalTableModel**: Uma especialização de QSqlTableModel mas com suporte a chaves estrangeiras (relações com outras tabelas).

Agenda

1 Parte Teórica

- Por que C++ e Qt?
- O que preciso para começar?

2 Prática - Construção de uma Agenda de Telefones

- Roteiro para a Prática
- Criando o projeto com o Qt Creator
- Trabalhando com Widgets, Layouts, Actions Etc.
- Conectando Signals e Slots
- Acesso a Bancos de Dados
- **Traduzindo a Aplicação**
- Construindo o instalador da Aplicação

Qt Linguist

A forma mais fácil de tornar uma aplicação Qt com suporte a outras línguas é:

- Usar o método `tr()` em todas as `QStrings` desta aplicação. Este método é definido em `QObject`.
- Carregar no momento de inicialização da aplicação um arquivo de tradução (`.qm`), previamente gerado com o `Qt Linguist`.

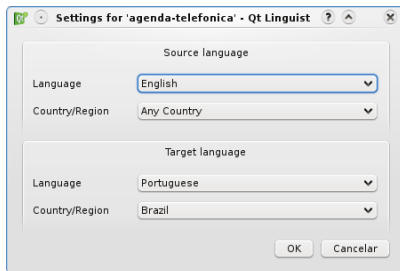
Produzindo um arquivo de tradução

Para preparar um arquivo de tradução deve-se usar a ferramenta **lupdate**, presente no diretório **bin** do Qt. Esta ferramenta é capaz de extrair todos os literais de strings presentes em janelas criadas com o Qt Designer e também encapsuladas por **tr()** nos demais arquivos fontes da aplicação.

Sintaxe: **lupdate** arquivo_do_projeto.pro

Produzindo um arquivo de tradução

Com a criação do arquivo fonte de tradução (.ts), basta apenas abri-lo com o Qt Linguist. Na primeira janela de diálogo, se definem as configurações das línguas de origem e destino dos textos capturados.



Produzindo um arquivo de tradução

O **Qt Linguist** é capaz de exibir visualmente a janela ou o código fonte no qual se está fazendo a tradução de uma string. Para cada string, realiza-se a tradução em uma seção como esta:



The image shows a window from the Qt Linguist application. It contains three sections:

- Source text**: A text area containing the word "All".
- Portuguese translation**: A text area containing the word "Todos".
- Portuguese translator comments**: An empty text area for additional notes.

Produzindo um arquivo de tradução

Após concluir todas as traduções das strings, utiliza-se a opção de menu **File->Release** do Qt Linguist. Com isto, será gerado um arquivo de compilação (.qm) do projeto de tradução. Para carregar este arquivo no momento de início da aplicação, pode-se utilizar o seguinte código no **main.cpp**:

```
QApplication a(argc, argv);  
QTranslator tradutor;  
tradutor.load("agenda-telefonica.qm",  
             QApplication::applicationDirPath());  
a.installTranslator(tradutor);
```

Agenda

- 1 Parte Teórica
 - Por que C++ e Qt?
 - O que preciso para começar?
- 2 **Prática - Construção de uma Agenda de Telefones**
 - Roteiro para a Prática
 - Criando o projeto com o Qt Creator
 - Trabalhando com Widgets, Layouts, Actions Etc.
 - Conectando Signals e Slots
 - Acesso a Bancos de Dados
 - Traduzindo a Aplicação
 - **Construindo o instalador da Aplicação**

Implantando uma aplicação Qt

Uma aplicação Qt (com compilação dinâmica) usualmente necessitará da seguinte estrutura:

- **Arquivo executável da aplicação.**
- **Arquivos .qm:** arquivos de tradução.
- **Arquivos de bibliotecas do Qt:** no linux, por exemplo, arquivos **libQtCore.so**, **libQtSql.so**, **libQtGui.so** (caso não se deseje utilizar as bibliotecas fornecidas na distribuição linux).
- **Diretório sqldrivers:** com os plugins do Qt de drivers para conexão a bancos de dados.
- **Arquivo .sh:** shell script que poderá configurar **variáveis de ambiente temporárias** e executar o **arquivo executável**.

Implantando uma aplicação Qt

No caso da aplicação exemplo, Agenda de Telefones:

- **Arquivo executável**: agenda-telefonica
- **Arquivos .qm**: agenda-telefonica.qm
- **Arquivos de bibliotecas do Qt**: libQtCore.so.4, libQtSql.so.4, libQtGui.so.4, libQtNetwork.so.4
- **Diretório sqldrivers**: com o plugin **libqsqlite.so**
- **Arquivo .sh**: agenda-telefonica.sh
- **Ícones**: agenda-telefonica.png e uninstall.png

Implantando uma aplicação Qt

Existem várias formas possíveis de construir um pacote de implantação de uma aplicação Qt. Algumas delas são:

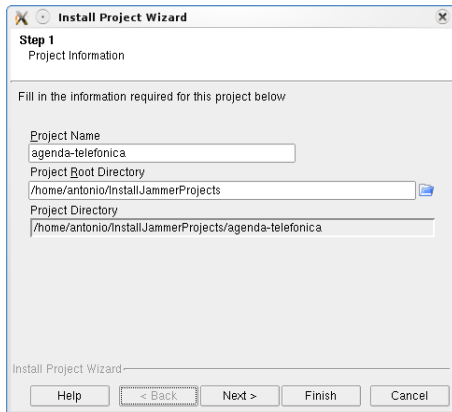
- **Pacotes .rpm, .deb, .tgz**: a criação de pacotes específicos de distribuições linux é uma das formas mais comuns e tende a transmitir um “quê” de formalidade no mundo linux.
- **Pacotes de código fonte**: também são muito utilizados, mas exigem um maior conhecimento dos usuários no momento de compilar e decidir onde implantar o software.
- **Instaladores executáveis**: tende a ser a opção mais simples para usuários e agrada bastante a quem já está acostumado com instaladores no Windows.

Implantando uma aplicação Qt

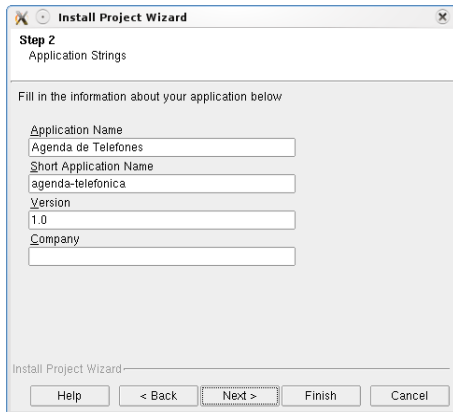
Aqui será tratada a implantação a partir de **instaladores executáveis**, mais precisamente criados com o **installJammer**.

Como um dos aspectos importantes em desenvolver com Qt é a capacidade de desenvolver sistemas **multiplataformas**, é interessante utilizar uma ferramenta de instalação **multiplataforma**. Esta é uma característica do installJammer.

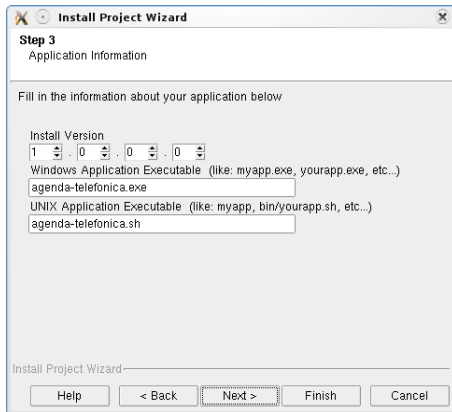
Criando o instalador



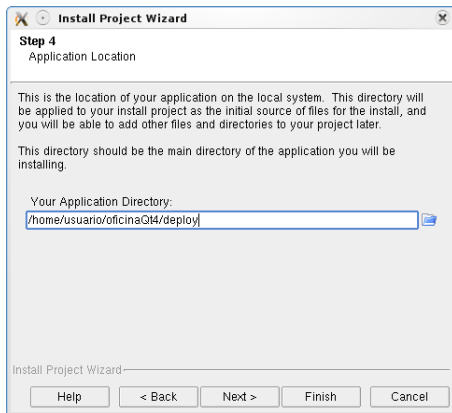
Criando o instalador



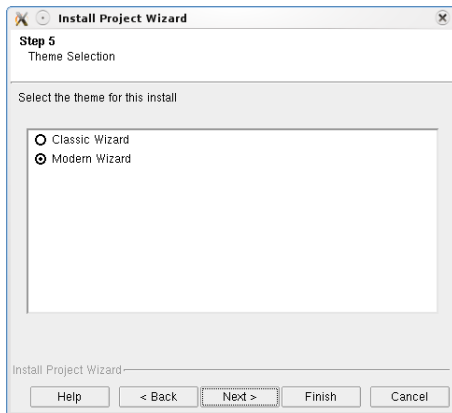
Criando o instalador



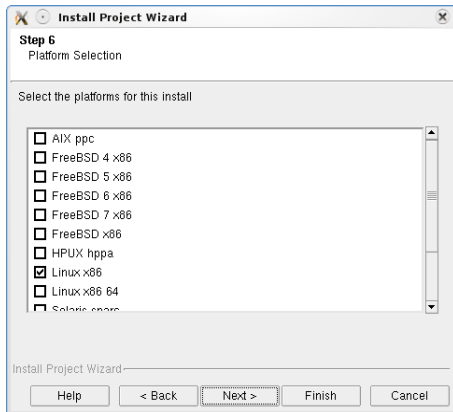
Criando o instalador



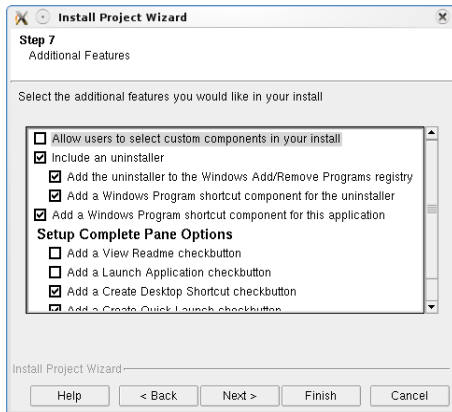
Criando o instalador



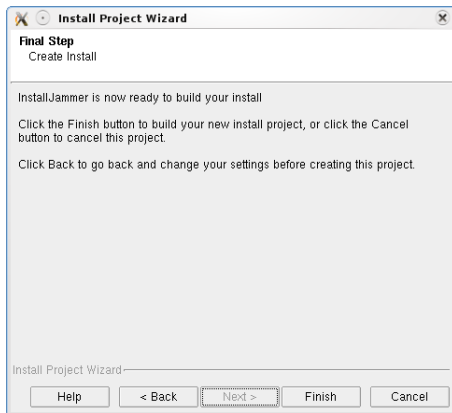
Criando o instalador



Criando o instalador



Criando o instalador



Criando o instalador

Após passar pelas telas do **assistente de criação de projeto** do InstallJammer, será exibida a tela com a árvore de configurações do projeto. Para configurar os ícones da aplicação, configure:

Em **Action Groups** → **InstallActions** :

ProgramShortcut → **IconPath** :

<%InstallDir%>agenda-telefonica.png

Uninstall Shortcut → **IconPath** :

<%InstallDir%>uninstall.png

Criando o instalador

Em **Action Groups** → **FinishActions** :

InstallDesktopShortcut → **IconPath** :

<%InstallDir%>agenda-telefonica.png

Install Quick Launch Shortcut → **IconPath** :

<%InstallDir%>agenda-telefonica.png

Para criar o instalador, basta executar a opção **Build Install**.

Resumo do que foi visto

- **Introdução ao Qt.**
 - Como iniciar um projeto de aplicação C++ com Qt.
 - Como realizar o projeto de telas com Qt Designer.
 - Como criar mecanismos de controle através de signals e slots.
 - Como trabalhar com banco de dados em uma aplicação Qt.
 - Como traduzir uma aplicação Qt para outra língua.
 - Como produzir o instalador da aplicação.

Resumo do que foi visto

- Introdução ao Qt.
- Como iniciar um projeto de aplicação C++ com Qt.
- Como realizar o projeto de telas com Qt Designer.
- Como criar mecanismos de controle através de signals e slots.
- Como trabalhar com banco de dados em uma aplicação Qt.
- Como traduzir uma aplicação Qt para outra língua.
- Como produzir o instalador da aplicação.

Resumo do que foi visto

- Introdução ao Qt.
- Como iniciar um projeto de aplicação C++ com Qt.
- Como realizar o projeto de telas com Qt Designer.
- Como criar mecanismos de controle através de signals e slots.
- Como trabalhar com banco de dados em uma aplicação Qt.
- Como traduzir uma aplicação Qt para outra língua.
- Como produzir o instalador da aplicação.

Resumo do que foi visto

- Introdução ao Qt.
- Como iniciar um projeto de aplicação C++ com Qt.
- Como realizar o projeto de telas com Qt Designer.
- Como criar mecanismos de controle através de signals e slots.
- Como trabalhar com banco de dados em uma aplicação Qt.
- Como traduzir uma aplicação Qt para outra língua.
- Como produzir o instalador da aplicação.

Resumo do que foi visto

- Introdução ao Qt.
- Como iniciar um projeto de aplicação C++ com Qt.
- Como realizar o projeto de telas com Qt Designer.
- Como criar mecanismos de controle através de signals e slots.
- Como trabalhar com banco de dados em uma aplicação Qt.
- Como traduzir uma aplicação Qt para outra língua.
- Como produzir o instalador da aplicação.

Resumo do que foi visto

- Introdução ao Qt.
- Como iniciar um projeto de aplicação C++ com Qt.
- Como realizar o projeto de telas com Qt Designer.
- Como criar mecanismos de controle através de signals e slots.
- Como trabalhar com banco de dados em uma aplicação Qt.
- Como traduzir uma aplicação Qt para outra língua.
- Como produzir o instalador da aplicação.





Resumo do que foi visto

- Introdução ao Qt.
- Como iniciar um projeto de aplicação C++ com Qt.
- Como realizar o projeto de telas com Qt Designer.
- Como criar mecanismos de controle através de signals e slots.
- Como trabalhar com banco de dados em uma aplicação Qt.
- Como traduzir uma aplicação Qt para outra língua.
- Como produzir o instalador da aplicação.

Resumo do que foi visto

- Introdução ao Qt.
- Como iniciar um projeto de aplicação C++ com Qt.
- Como realizar o projeto de telas com Qt Designer.
- Como criar mecanismos de controle através de signals e slots.
- Como trabalhar com banco de dados em uma aplicação Qt.
- Como traduzir uma aplicação Qt para outra língua.
- Como produzir o instalador da aplicação.

Referências

-  J. Blanchette and M. Summerfield.
C++ GUI Programming with Qt 4.
Prentice Hall, 2008.
-  Documentação do Qt 4.5
<http://qt.nokia.com/doc/4.5/index.html>
-  Página do InstallJammer
<http://www.installjammer.com>
-  IDE Kevora
<http://kevora.sourceforge.net>