QtCon Brasil 2018

Desenvolvendo aplicações com Qt em Linux embarcado

Embedded **Labworks**

Por Sergio Prado. São Paulo, Novembro de 2018

SOBRE ESTE DOCUMENTO

- Os fontes deste documento estão disponíveis em:
  http://e-labworks.com/palestras/qtconbrasil2018

- Sinta-se livre para utilizar, compartilhar e adaptar este material às suas necessidades, lembrando-se sempre de respeitar a licença.

SERGIO PRADO

- Sergio Prado tem mais de 20 anos de experiência em desenvolvimento de software para sistemas embarcados, em diversas arquiteturas de CPU (ARM, PPC, MIPS, x86, etc), atuando em projetos com Linux embarcado, Android embarcado e sistemas operacionais de tempo real.

- É sócio da Embedded Labworks, onde atua com consultoria, treinamento e desenvolvimento de software para sistemas embarcados.

- É ativo na comunidade de sistemas embarcados no Brasil, sendo um dos criadores do site Embarcados, administrador do grupo sis_embarcados no Google Groups, além de manter um blog pessoal sobre assuntos da área.

  http://sergioprado.org

- É colaborador de alguns projetos de software livre, incluindo o Buildroot e o kernel Linux.

AGENDA (PARTE 1)

1. Criando distribuições GNU/Linux com suporte ao Qt5 para dispositivos embarcados.

2. Configurando o Qt Creator para desenvolver e depurar aplicações em Qt5 para um dispositivo com Linux embarcado.

PRÉ-REQUISITOS

- Usuário de sistemas operacionais GNU/Linux.
- Terminal de comandos (ls, cat, cp, mv, grep, find, vi, etc).
- Conhecimentos básicos de linguagem C e C++.

AMBIENTE DE LABORATÓRIO

```
/opt/labs/                  Ambiente de laboratório
    dl/                     Aplicações e pacotes open-source
                            que serão utilizados durante as
                            atividades de laboratório
    docs/                   Documentação
        hardware/           Documentação do hardware
        training-part1/     Atividades de laboratório (parte 1)
        training-part2/     Atividades de laboratório (parte 2)
    ex/                     Exercícios de laboratório
```

ORIENTAÇÕES GERAIS

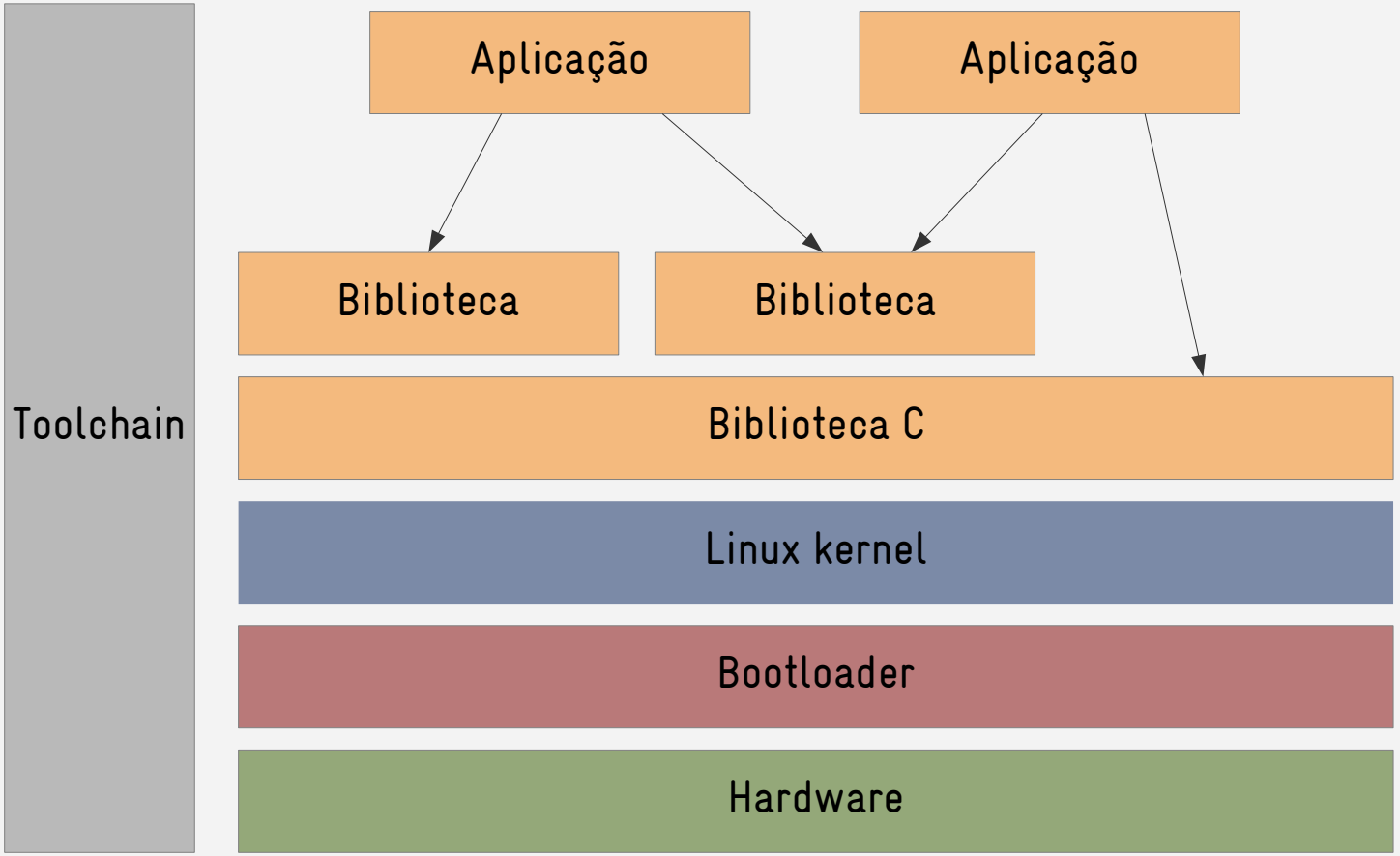
- Pergunte...
- Expresse seu ponto de vista...
- Troque experiências...
- Ajude...
- Participe!

Embedded Labworks

QtCon Brasil 2018

Construindo distribuições GNU/Linux

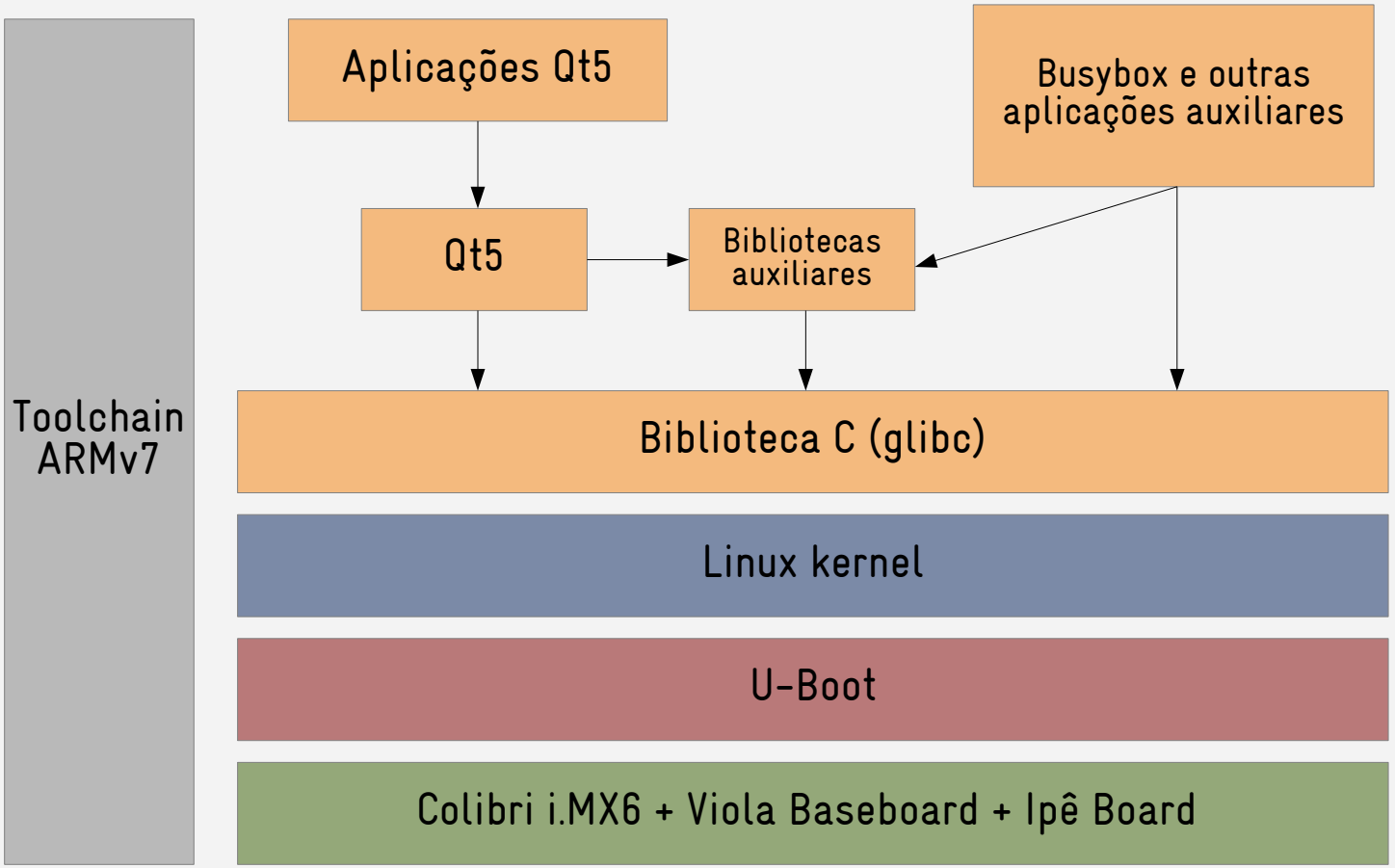
SISTEMA LINUX EMBARCADO



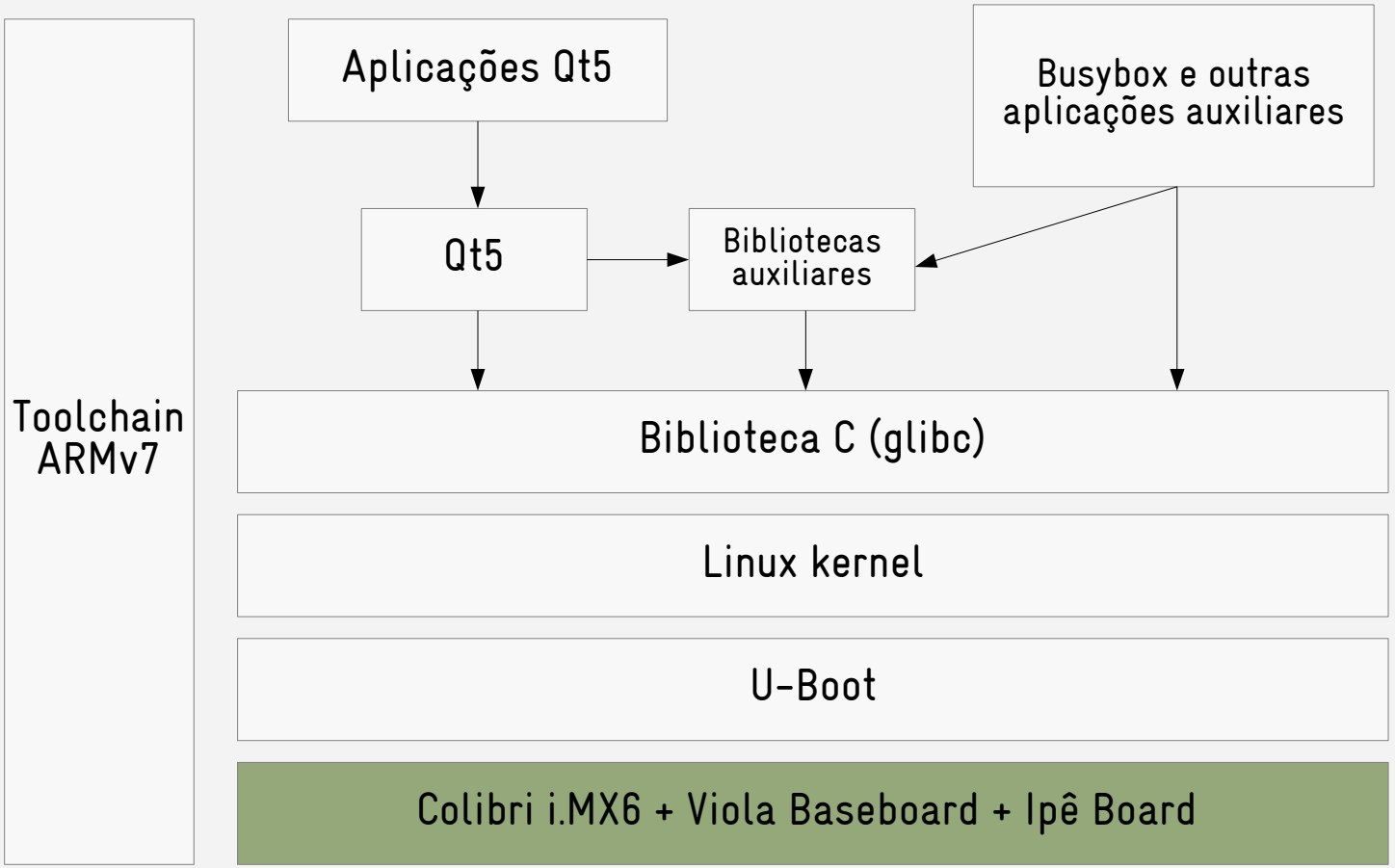
COMPONENTES DE UM SISTEMA LINUX

- **Hardware**: dispositivo de hardware (target).
- **Bootloader**: responsável pela inicialização básica do hardware, carga e execução do sistema operacional, no nosso caso o kernel Linux.
- **Kernel Linux**: Núcleo do sistema operacional. Gerencia CPU, memória e I/O, exportando diversos serviços para a camada de usuário.
- **Rootfs**: sistema de arquivos principal.
  - Biblioteca C: interface entre o kernel Linux e as aplicações do usuário.
  - Bibliotecas e aplicações do usuário.
- **Toolchain**: conjunto de ferramentas para gerar os binários do sistema.

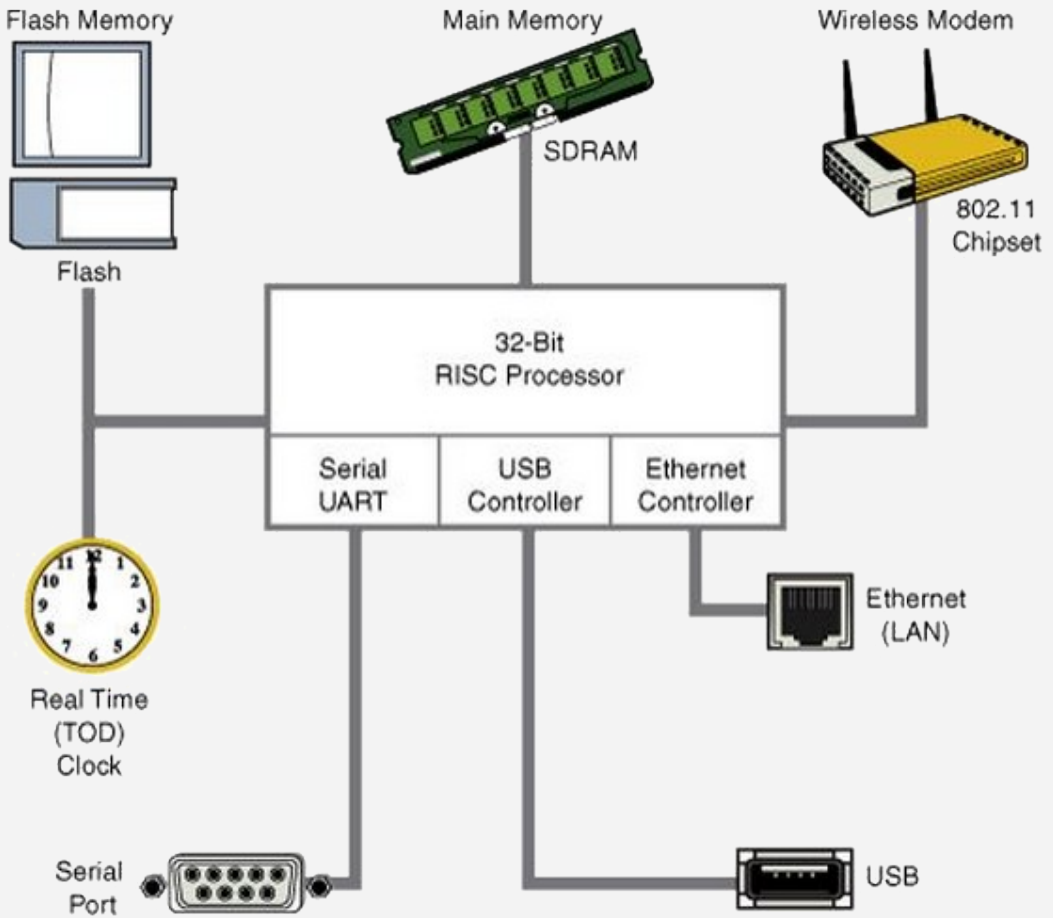
SISTEMA LINUX COM O QT5



HARDWARE


Toolchain ARMv7
Aplicações Qt5
Busybox e outras aplicações auxiliares
Qt5
Bibliotecas auxiliares
Biblioteca C (glibc)
Linux kernel
U-Boot
Colibri i.MX6 + Viola Baseboard + Ipê Board

HARDWARE (cont.)



CPU

- Suporta mais de 30 arquiteturas diferentes (x86, ia64, ARM, PPC, MIPS, SuperH, Blackfin, Coldfire, etc).
- 32/64 bits: não foi feito para microcontroladores!
- Originalmente projetado para CPUs com MMU (Memory Management Unit). O projeto uClinux foi criado para que o Linux pudesse ser usado em CPUs sem MMU.

  http://www.uclinux.org/
- Boa parte do uClinux já foi integrado à árvore oficial do kernel, possibilitando o uso do Linux em diversas CPUs sem MMU (m68k e arm sem MMU, H8/300 da Hitachi, ADI Blackfin, etc).

MEMÓRIA RAM E ARMAZENAMENTO

- Um sistema Linux bem básico pode funcionar com 8MB de RAM (ou até menos), mas o ideal para começar é em torno de 32MB.
- Suporta armazenamento em memória flash NAND e NOR, disco rígido, cartão SD/MMC, etc.
- Um sistema bem básico pode funcionar com 2M de armazenamento (ou até menos).
- Atualmente boa parte das plataformas de hardware utilizam a eMMC como dispositivo de armazenamento.

COMUNICAÇÃO

- O Linux suporta muitos barramentos comuns em sistemas embarcados: I2C, SPI, CAN, 1-wire, SDIO, USB, etc.
- E também os principais protocolos de rede: Ethernet, Wi-Fi, Bluetooth, CAN, IPv4, IPv6, TCP, UDP, etc.
- Se o barramento ou protocolo não possuir restrições de licença, é bem provável que esteja implementado no kernel.
- Já protocolos ou barramentos com restrições de licença tem dificuldade para entrar na árvore oficial do kernel (ex: Zigbee).

CRITÉRIOS PARA SELEÇÃO

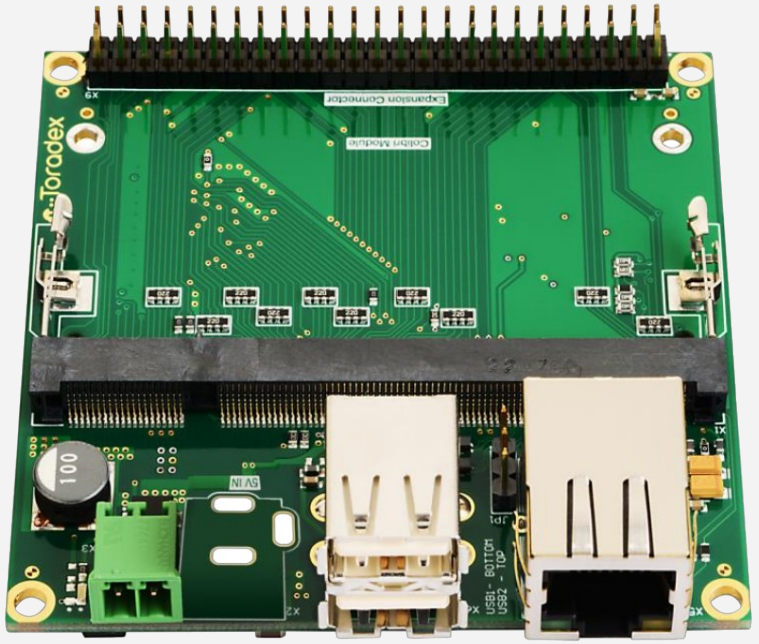
- Certifique-se de que o hardware já é suportado pelo Linux e por um bootloader open-source.
- Suporte nas versões oficiais dos projetos (bootloader e kernel) é melhor: maior qualidade e novas versões disponíveis.
- A diferença entre uma plataforma suportada na árvore oficial do kernel, e outra plataforma não suportada de forma oficial, pode te trazer grandes consequências em termos de custo e tempo de desenvolvimento!

COLIBRI IMX6DL

- System-on-module (SOM) da Toradex.
- Baseado no SOC i.MX6 DualLite da NXP, um ARM Cortex-A9 rodando a até 996MHz.
- 512MB de memória RAM.
- 4GB de armazenamento interno (eMMC NAND Flash).
- Conector no padrão SODIMM200 (memória DDR1).

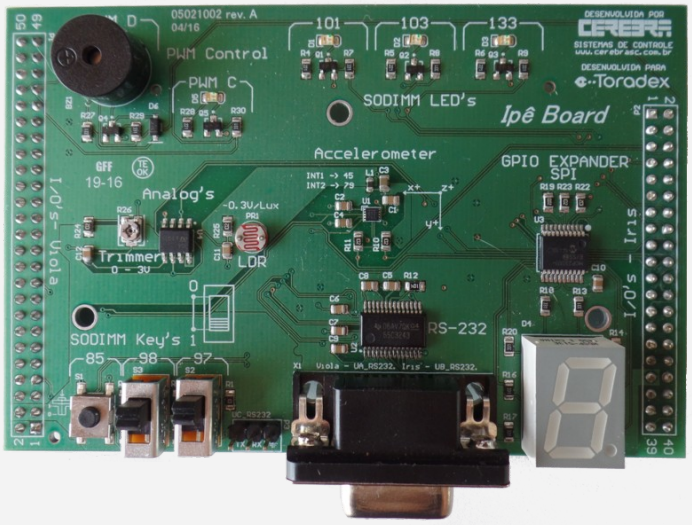
PLACA BASE VIOLA

- Placa base compatível com todos os módulos Colibri da Toradex.
- 2 conectores USB Host e 1 conector Ethernet.
- 1 conector de cartão SD.
- Interface RGB para display LCD.
- Barramento de 50 pinos que exporta o acesso às diversas interfaces de I/O do SOC (I2C, SPI, UART, GPIO, etc).
- Alimentação externa com uma fonte de 5V/2A.



PLACA DE EXPANSÃO IPÊ

- Placa desenvolvida pela Toradex Brasil.
- 2 chaves, 1 botão e 3 leds conectados a GPIOs.
- 1 led e 1 buzzer conectados a canais PWM.
- 1 porta serial para console.
- 1 resistor dependente de luz (LDR) e um trimpot conectados a canais A/D.
- 1 acelerômetro MMA8653.
- 1 expansor de GPIOs MCP23S08 conectado a um display de 7 segmentos.



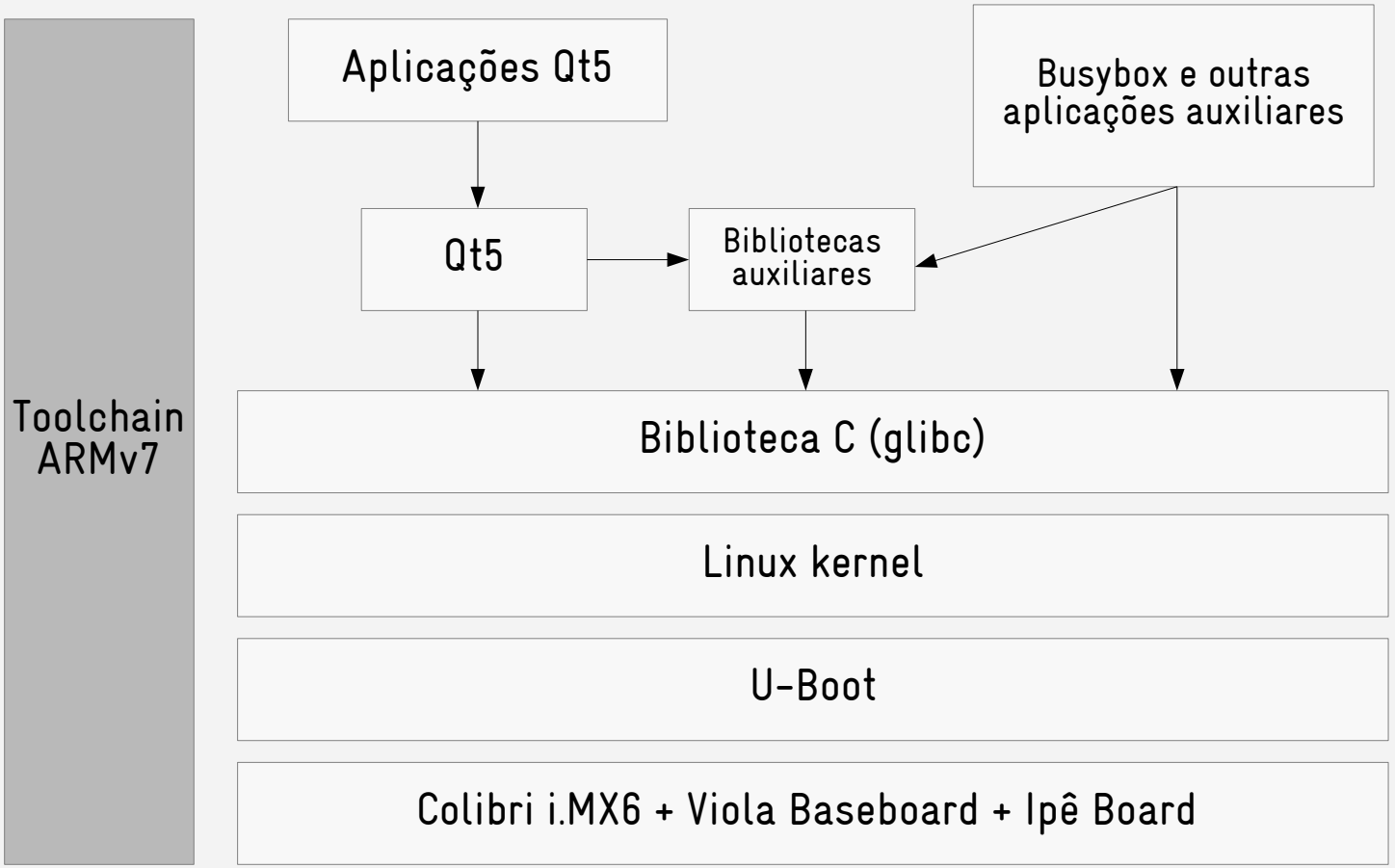
REFERÊNCIAS E DOCUMENTAÇÃO

- A documentação do hardware está disponível no ambiente de laboratório do treinamento em `/opt/labs/docs/hardware`:
  - `SOC_imx6sdl_datasheet.pdf`: datasheet do SOC.
  - `SOM_colibri_imx6_datasheet.pdf`: datasheet do SOM.
  - `BASE_BOARD_viola_datasheet.pdf`: datasheet da placa base.
  - `EXT_BOARD_ipe_esquemático.pdf`: esquemático da placa de expansão.

- Recursos na internet:

  http://www.toradex.com/
  https://www.toradex.com/community/
  http://community.nxp.com

TOOLCHAIN


Toolchain ARMv7
Aplicações Qt5
Busybox e outras aplicações auxiliares
Qt5
Bibliotecas auxiliares
Biblioteca C (glibc)
Linux kernel
U-Boot
Colibri i.MX6 + Viola Baseboard + Ipê Board

O QUE SÃO TOOLCHAINS?

- Ao pé da letra, e traduzindo literalmente, toolchain é uma "corrente de ferramentas". Na prática, é um conjunto de ferramentas de compilação.

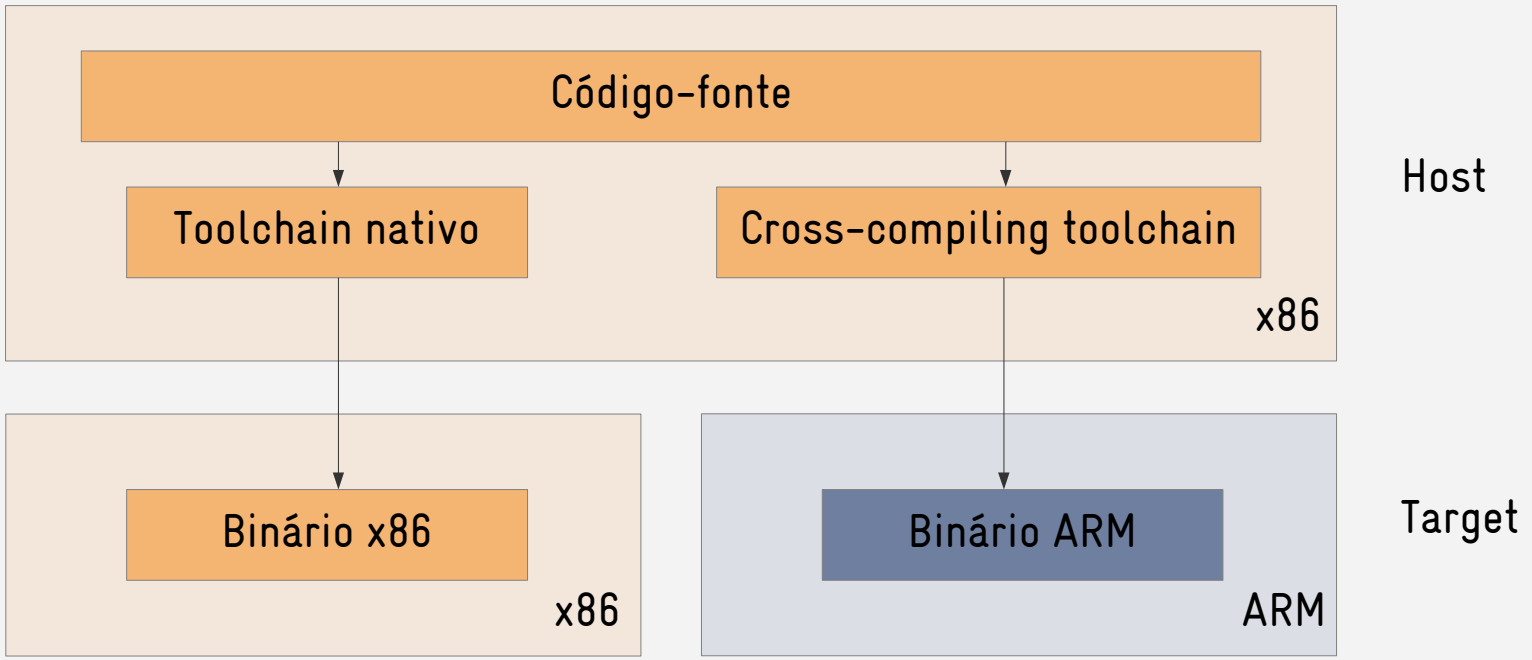
- Você se lembra do processo de compilação de um código em C? Ele envolve normalmente as seguintes etapas: pré-processamento, compilação, montagem (assembler) e linkagem.

- Cada uma destas etapas é executada por uma ferramenta (pré-processador, compilador, assembler e linker), e todas elas fazem parte do toolchain.

TIPOS DE TOOLCHAIN

- As ferramentas de desenvolvimento normalmente disponíveis em um desktop GNU/Linux são chamadas de **toolchain nativo**.
- Este toolchain roda na sua máquina e compila código para ser executado na sua máquina, geralmente um x86.
- Em desenvolvimento de sistemas embarcados normalmente é complicado (às vezes até impossível) usar um toolchain nativo, porque precisamos de bastante espaço em disco, capacidade de processamento, memória, etc.
- Portanto, para esta tarefa, o melhor é usar um **cross-compiling toolchain**, que roda na sua plataforma de desenvolvimento mas gera código para a sua plataforma alvo.

CROSS-COMPILING TOOLCHAIN



TOOLCHAIN BASEADO NO GNU

- **gcc**: compilador, com suporte a diversas linguagens como C, C++ e Fortran.

  http://gcc.gnu.org/

- **binutils**: ferramentas de manipulação de binários como o assembler e o linker.

  http://www.gnu.org/software/binutils/

- **glibc**: biblioteca C padrão do sistema.

  http://www.gnu.org/software/libc/

INSTALANDO UM TOOLCHAIN

- Existem alguns toolchains prontos disponíveis na Internet, como por exemplo o toolchain da Linaro.

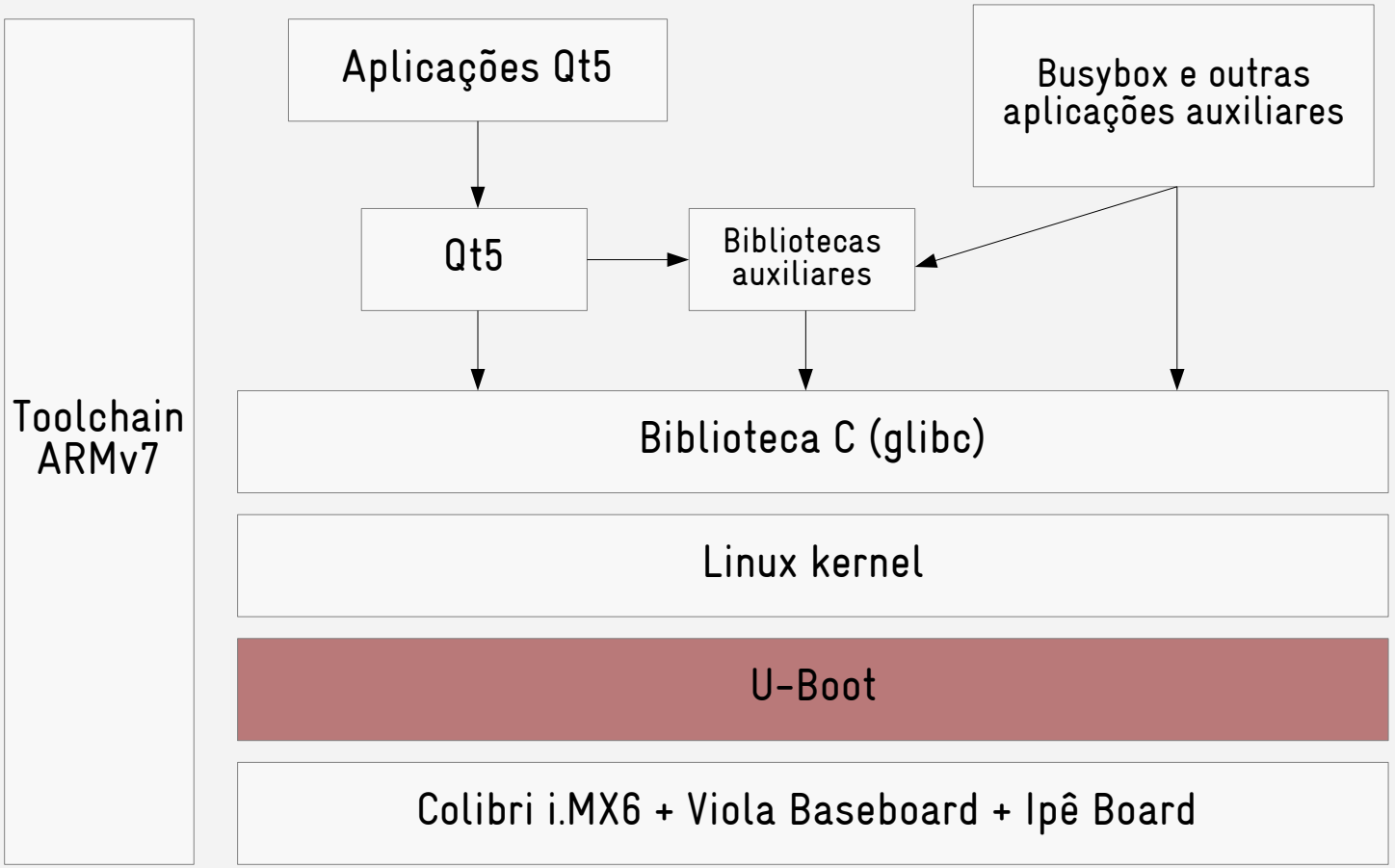
  https://wiki.linaro.org/WorkingGroups/ToolChain

- Uma distribuição GNU/Linux pode conter toolchains em seu repositório de pacotes:

  ```
  $ sudo apt-get install gcc-arm-linux-gnueabihf
  ```

- Existem algumas ferramentas capazes de gerar toolchains, incluindo o crosstool-ng, Buildroot e Yocto Project.

BOOTLOADER

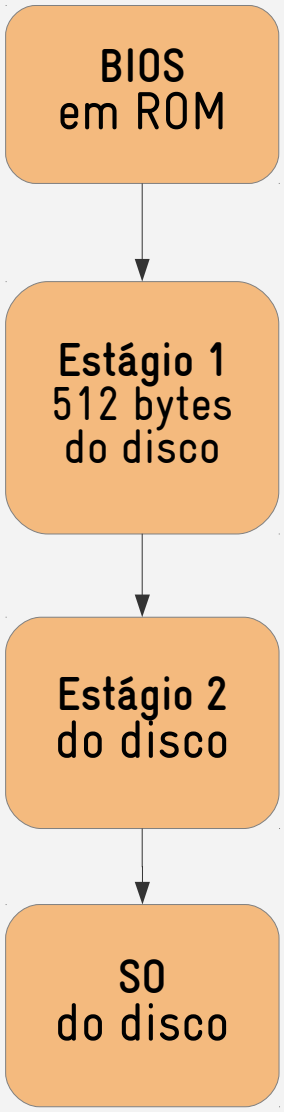

Toolchain ARMv7
Aplicações Qt5
Busybox e outras aplicações auxiliares
Qt5
Bibliotecas auxiliares
Biblioteca C (glibc)
Linux kernel
U-Boot
Colibri i.MX6 + Viola Baseboard + Ipê Board

BOOTLOADER (cont.)

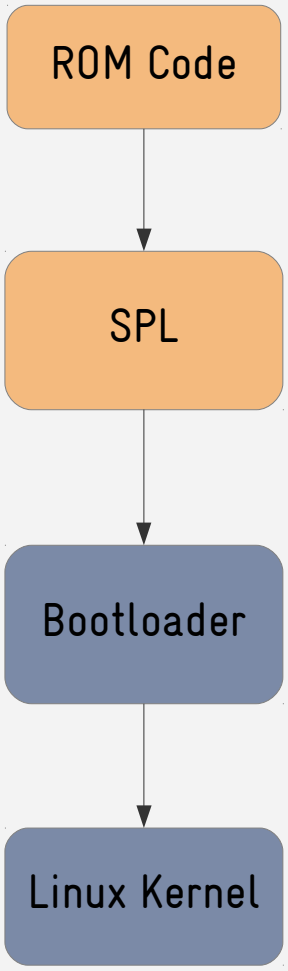
- O bootloader tem basicamente duas responsabilidades:
  - Inicializar o hardware.
  - Carregar e executar o sistema operacional.
- Normalmente o bootloader provê outras funcionalidades para facilitar o desenvolvimento do sistema, incluindo:
  - Fazer o boot pela rede ou pela porta serial.
  - Ler e escrever na memória flash.
  - Executar rotinas de diagnóstico de hardware.

BOOT EM X86

- Plataformas x86 normalmente vem acompanhadas de uma memória não-volátil, a BIOS.
- Um programa na BIOS é executado no boot do equipamento, que faz a inicialização básica do hardware, carrega para a memória e executa os primeiros 512 bytes do dispositivo de boot. Estes 512 bytes também são chamados de MBR.
- A MBR é o bootloader de 1o. estágio, que é o responsável por carregar um bootloader de 2o. estágio do disco para a RAM.
- O bootloader de 2o. estágio é mais completo, entende sistemas de arquivo, consegue ler o sistema operacional do disco, carregar para a memória e executar.



BOOT EM ARM



O SoC tem um código de boot em uma ROM interna, responsável por carregar um bootloader de 1o. estágio (SPL) para uma memória RAM interna (SRAM ou IRAM).

O SPL (Secundary Program Loader) é responsável por inicializar o hardware (CPU, DRAM, GPIOs, etc) e carregar um bootloader de 2o. estágio para a RAM.

É um bootloader mais completo e normalmente suporta sistemas de arquivo, interface USB e protocolo TCP/IP. Responsável por carregar e executar o kernel Linux.

É executado da memória RAM e assume o controle do sistema (a partir daqui, o bootloader não existe mais).

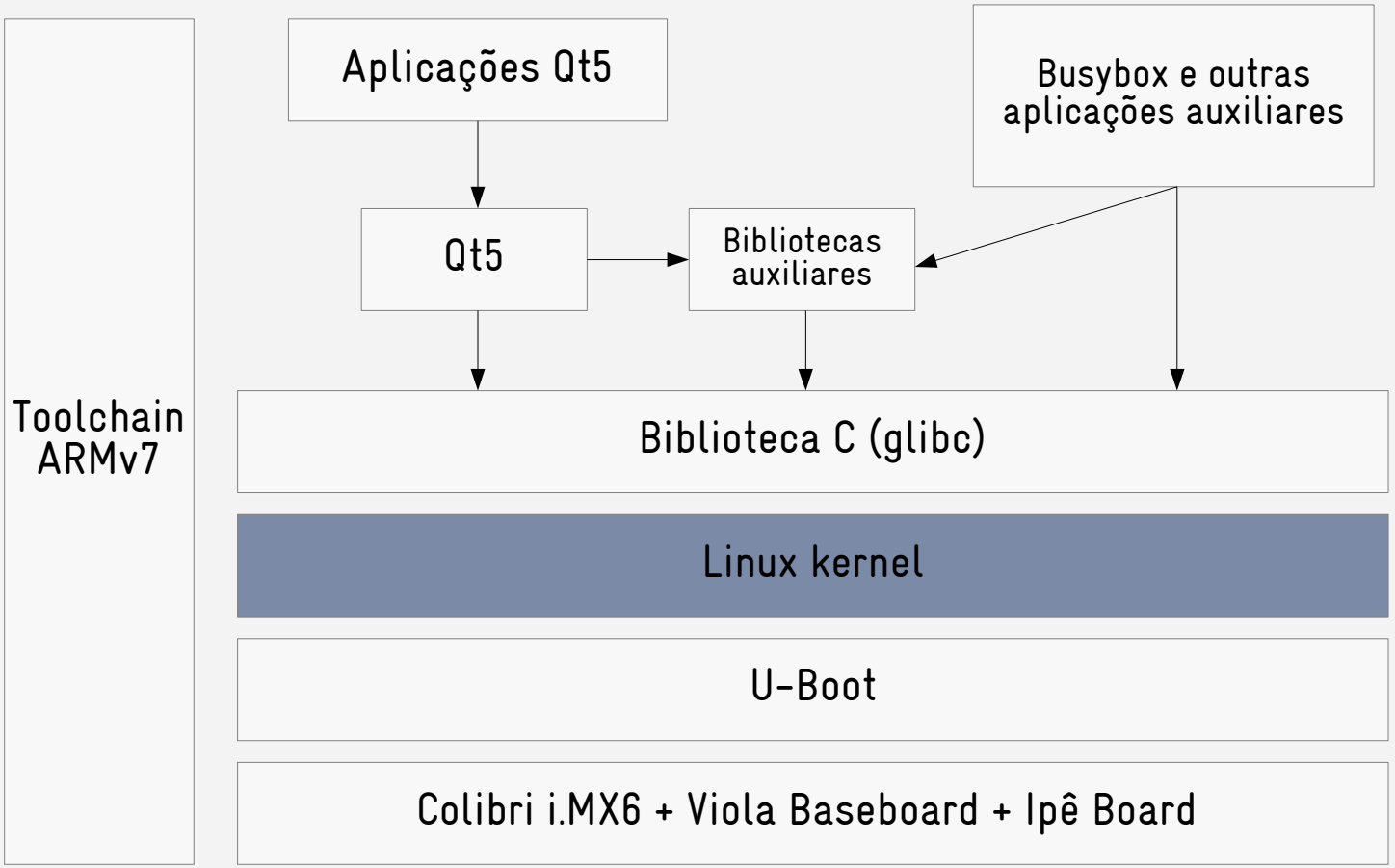
PRINCIPAIS BOOTLOADERS

- x86:
  - LILO
  - Grub
  - Syslinux
- ARM, MIPS, PPC e outras arquiteturas:
  - U-Boot
  - Barebox
  - Redboot

U-BOOT

- Bootloader open-source (GPLv2) mais utilizado atualmente, principalmente em ARM.
  http://www.denx.de/wiki/U-Boot
- Suporta uma grande variedade de CPUs, incluindo PPC, ARM, MIPS, Coldfire, x86, etc.
- Documentação disponível no site do projeto.
  http://www.denx.de/wiki/U-Boot/Documentation

KERNEL LINUX



KERNEL LINUX (cont.)

- O Linux é um kernel!
  https://kernel.org
- As distribuições GNU/Linux (Ubuntu, Fedora, Debian, Slackware, etc) integram o kernel Linux, bibliotecas e aplicações.
- Criado em 1991 pelo estudante finlandês Linus Torvalds, começou a ser usado rapidamente como sistema operacional em projetos de software livre.
- Linus Torvalds foi capaz de criar uma comunidade grande e dinâmica de desenvolvedores e usuários ao redor do projeto. Atualmente, centenas de pessoas e empresas contribuem com o projeto.

KERNEL LINUX (cont.)

Most active 4.2 employers

By changesets			By lines changed		
Intel	1665	12.3%	AMD	438094	36.8%
Red Hat	1639	12.1%	Intel	96331	8.1%
(Unknown)	884	6.5%	Red Hat	62959	5.3%
(None)	884	6.5%	(None)	46140	3.9%
Samsung	681	5.0%	(Unknown)	41886	3.5%
SUSE	496	3.7%	Atmel	34942	2.9%
Linaro	449	3.3%	Samsung	29326	2.5%
(Consultant)	412	3.0%	Linaro	22714	1.9%
IBM	391	2.9%	Cisco	21170	1.8%
AMD	286	2.1%	SUSE	18891	1.6%
Google	246	1.8%	Code Aurora Forum	18435	1.5%
Renesas Electronics	203	1.5%	Mellanox	18044	1.5%
Free Electrons	203	1.5%	(Consultant)	15234	1.3%
Texas Instruments	191	1.4%	IBM	15095	1.3%
Facebook	176	1.3%	Cavium Networks	14580	1.2%
Oracle	163	1.2%	Free Electrons	13640	1.1%
Freescale	156	1.2%	Unisys	13428	1.1%
ARM	145	1.1%	Linux Foundation	12617	1.1%
Cisco	142	1.0%	MediaTek	11856	1.0%
Broadcom	138	1.0%	Google	11811	1.0%

KERNEL LINUX (cont.)

- O kernel Linux abstrai o uso das CPUs do sistema, de forma que cada processo acredite que ele tem a CPU só para ele.
- O kernel Linux abstrai o uso da memória com a ajuda da MMU, de forma que cada processo acredite que ele tem a memória só para ele.
- O kernel Linux abstrai o acesso a dispositivos de I/O, utilizando arquivos como principal mecanismo de abstração.

```
$ echo “hello” > /dev/ttyS0
```

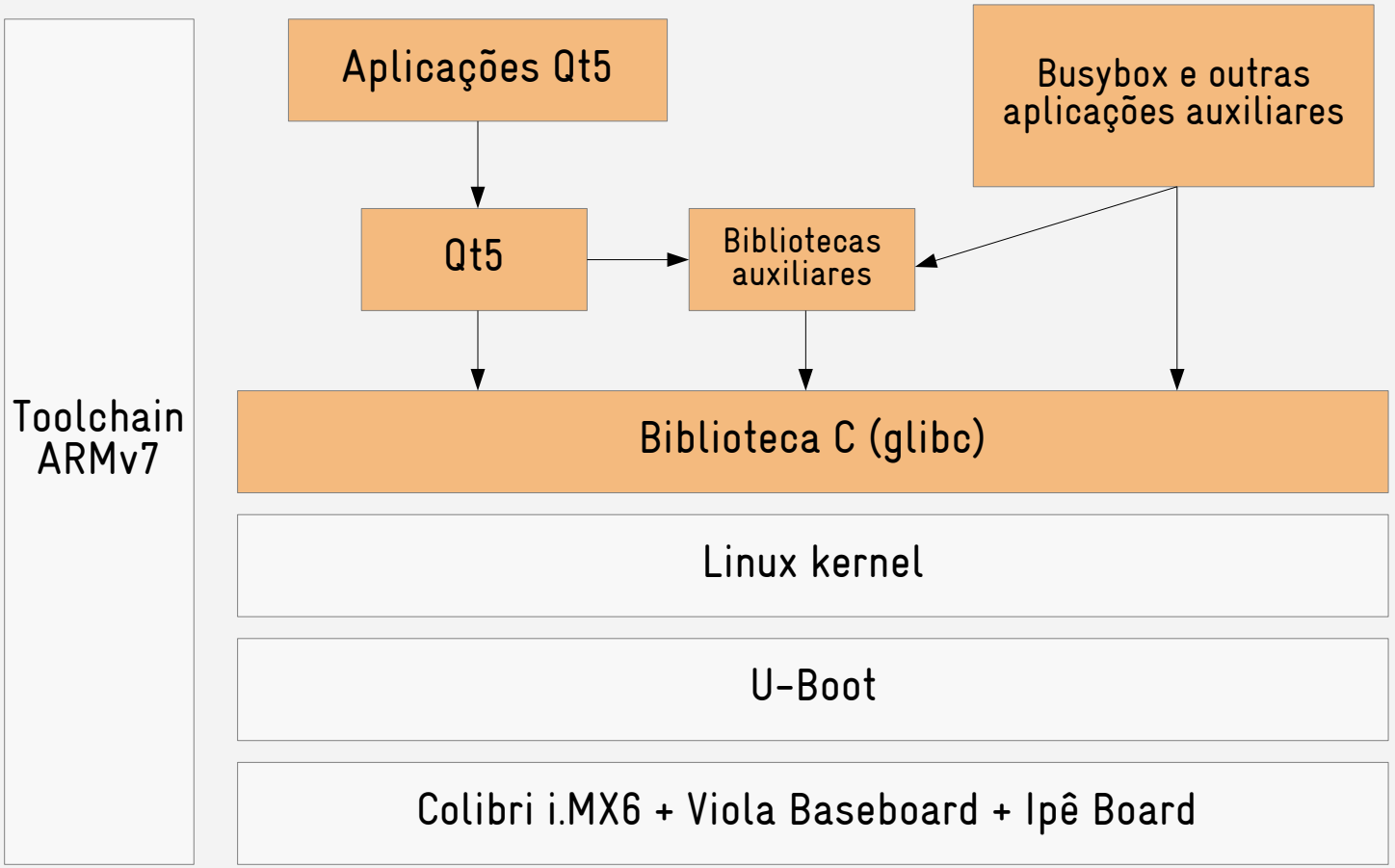
KERNEL LINUX (cont.)

- Existe uma separação bem definida entre o kernel (kernel space) e as bibliotecas e aplicações do usuário (user space).
- O kernel roda em modo privilegiado, com acesso completo a todas as instruções da CPU, endereçamento de memória e I/O, enquanto que os processos do usuário rodam em modo restrito, com acesso limitado aos recursos da máquina.
- Por isso, existe uma interface de comunicação, baseada chamadas de sistema (system calls), para que as bibliotecas e aplicações tenham acesso aos recursos da máquina.

DEVICE TREE

- Muitas plataformas possuem dispositivos de hardware que não podem ser identificados dinamicamente pelo kernel.
- Nestes casos, é necessário um mecanismo para comunicar ao kernel informações sobre os dispositivos de hardware presentes no sistema.
- Para resolver este problema, muitas plataformas adotaram o device tree.
- O device tree é uma estrutura de dados capaz de descrever a topologia e a configuração do hardware presente no sistema.
- Na prática, é um arquivo com extensão `.dts`, compilado para um arquivo com extensão `.dtb` e passado para o kernel no boot do sistema.

ROOTFS


Aplicações Qt5
Busybox e outras aplicações auxiliares
Qt5
Bibliotecas auxiliares
Toolchain ARMv7
Biblioteca C (glibc)
Linux kernel
U-Boot
Colibri i.MX6 + Viola Baseboard + Ipê Board

COMPONENTES BÁSICOS DO ROOTFS

- Um sistema GNU/Linux precisa de um conjunto básico de programas para funcionar, incluindo:
  - Uma biblioteca do sistema (glibc, uClibc-ng, musl, etc).
  - Um mecanismo de inicialização (systemd, sysvinit, upstart, etc).
  - Diversas bibliotecas e aplicações (bash, cat, echo, grep, sed, useradd, vi, getty, libusb, etc).
- Normalmente estes programas são fornecidos em diferentes projetos e é trabalhoso configurar, compilar e integrar manualmente todos eles.

BUSYBOX

- O **Busybox** é uma solução alternativa, trazendo uma quantidade grande e comum de programas usados em sistemas Linux, mas com tamanho reduzido, perfeito para sistemas embarcados!

  http://www.busybox.net/

- O Busybox contém diversos componentes, incluindo um sistema de inicialização baseado no sysvinit, um terminal de comandos, além de ferramentas e utilitários diversos (cat, echo, ps, vi, etc).

- Geralmente, as ferramentas são mais limitadas em termos de funcionalidades comparadas às originais.

BUSYBOX – TUDO ISSO EM ~1MB!

addgroup, **adduser**, adjtimex, ar, **arp**, arping, ash, awk, basename, bbconfig, bbsh, brctl, bunzip2, busybox, bzcat, **bzip2**, cal, cat, catv, chat, chattr, chcon, chgrp, chmod, chown, chpasswd, chpst, chroot, chrt, chvt, cksum, clear, cmp, comm, cp, **cpio**, crond, crontab, cryptpw, cttyhack, cut, date, dc, dd, deallocvt, delgroup, deluser, depmod, devfsd, df, dhcprelay, **diff**, dirname, dmesg, dnsd, dos2unix, dpkg, dpkg_deb, du, dumpkmap, dumpleases, **e2fsck**, echo, ed, egrep, eject, env, envdir, envuidgid, ether_wake, expand, expr, fakeidentd, false, fbset, fbsplash, fdflush, fdformat, fdisk, fetchmail, fgrep, **find**, findfs, fold, free, freeramdisk, fsck, fsck_minix, ftpget, ftpput, fuser, getenforce, getopt, getsebool, **getty**, grep, gunzip, gzip, halt, hd, hdparm, head, hexdump, hostid, hostname, **httpd**, hush, hwclock, id, ifconfig, ifdown, ifenslave, ifup, **inetd**, init, inotifyd, insmod, install, **ip**, ipaddr, ipcalc, ipcrm, ipcs, iplink, iproute, iprule, iptunnel, kbd_mode, **kill**, killall, killall5, klogd, lash, last, length, less, linux32, linux64, linuxrc, ln, load_policy, loadfont, loadkmap, logger, login, logname, logread, losetup, lpd, lpq, lpr, ls, lsattr, **lsmod**, lzmacat, makedevs, man, matchpathcon, md5sum, mdev, mesg, microcom, mkdir, mke2fs, mkfifo, mkfs_minix, mknod, mkswap, mktemp, **modprobe**, more, mount, mountpoint, msh, mt, mv, nameif, nc, **netstat**, nice, nmeter, nohup, nslookup, od, openvt, parse, **passwd**, patch, pgrep, pidof, ping, ping6, pipe_progress, pivot_root, pkill, poweroff, printenv, printf, **ps**, pscan, pwd, raidautorun, rdate, rdev, readahead, readlink, readprofile, realpath, reboot, renice, reset, resize, restorecon, rm, rmdir, rmmod, route, rpm, rpm2cpio, rtcwake, run_parts, runcon, runlevel, runsv, runsvdir, rx, script, sed, selinuxenabled, **sendmail**, seq, sestatus, setarch, setconsole, setenforce, setfiles, setfont, setkeycodes, setlogcons, setsebool, setsid, setuidgid, sh, sha1sum, showkey, slattach, sleep, softlimit, sort, split, start_stop_daemon, stat, strings, stty, su, sulogin, sum, sv, svlogd, swapoff, swapon, switch_root, sync, sysctl, syslogd, tac, tail, tar, taskset, tcpsvd, tee, telnet, **telnetd**, test, tftp, **tftpd**, time, top, touch, tr, traceroute, true, tty, ttysize, tune2fs, udhcpc, **udhcpd**, udpsvd, umount, uname, uncompress, unexpand, uniq, unix2dos, unlzma, unzip, uptime, usleep, uudecode, uuencode, vconfig, **vi**, vlock, watch, watchdog, wc, wget, which, who, whoami, xargs, yes, zcat, zcip

CRIANDO O ROOTFS

- No rootfs do sistema GNU/Linux que criaremos neste treinamento, incluiremos a glibc, as ferramentas básicas providas pelo Busybox, as bibliotecas do Qt5 e algumas bibliotecas e ferramentas auxiliares.

- Apesar de simples, criar um rootfs manualmente é bastante trabalhoso pois envolve configurar e compilar cada componente individualmente.

- Esta tarefa pode ser automatizada através de um sistema de build (build system).

BUILD SYSTEM

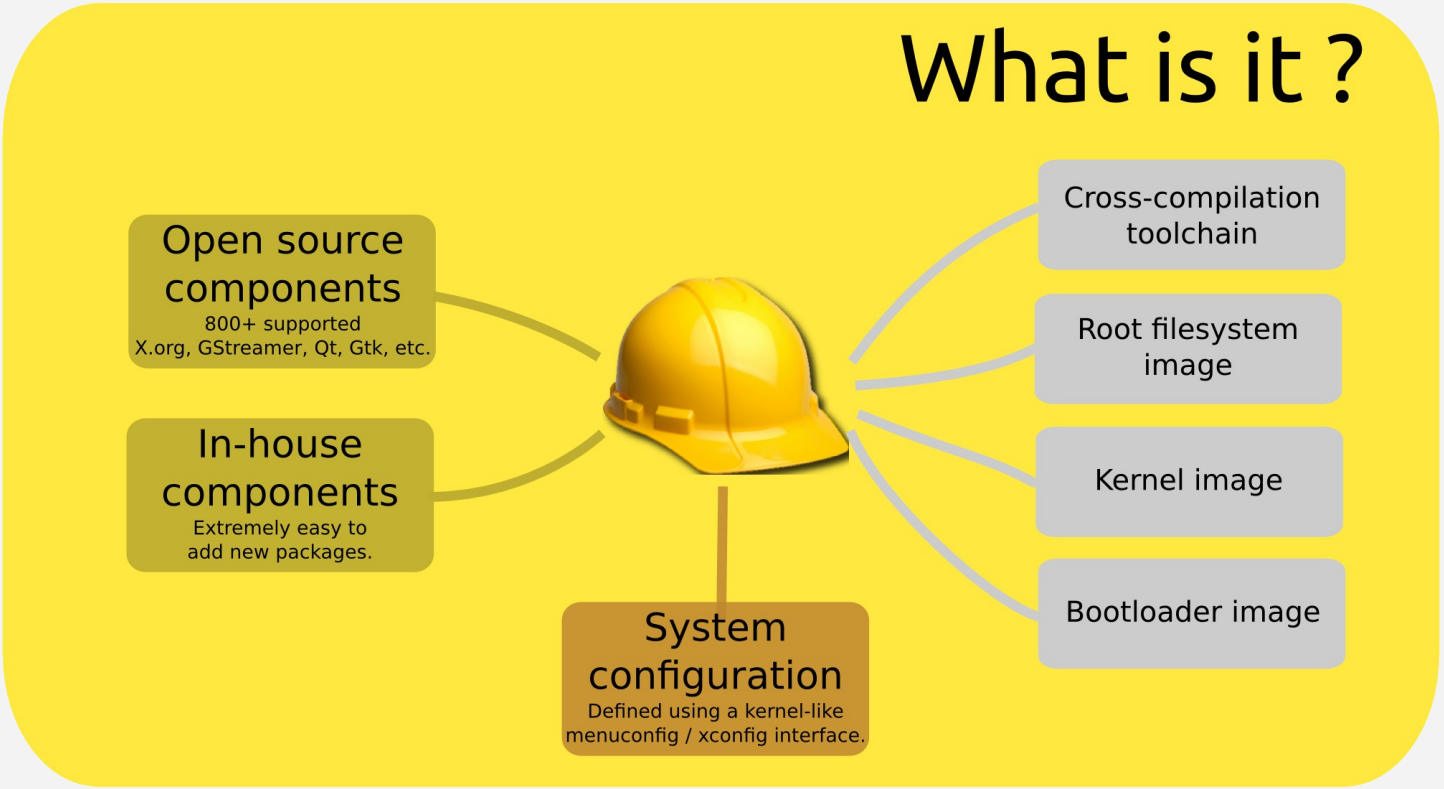
- Um build system é capaz de gerar todos os componentes do sistema operacional, incluindo o toolchain, bootloader, kernel Linux e rootfs.
- Atualmente, os dois principais sistemas de build são o Buildroot e o Yocto Project.
- Nesta apresentação, utilizaremos o Buildroot para construir a imagem do sistema operacional.

  http://www.buildroot.net

BUILDROOT

- Desenvolvido pelos mesmos mantenedores da uClibc.
- Possibilita gerar o toolchain, o bootloader, o kernel e o rootfs com muitas bibliotecas e aplicações disponíveis.
- Mais de 2.000 aplicações e bibliotecas integradas, de utilitários básicos à bibliotecas mais elaboradas como X.org, Qt, Gtk, Webkit, Gstreamer, etc.
- Desde a versão 2009.02 um novo release é liberado a cada 3 meses.

BUILDROOT (cont.)



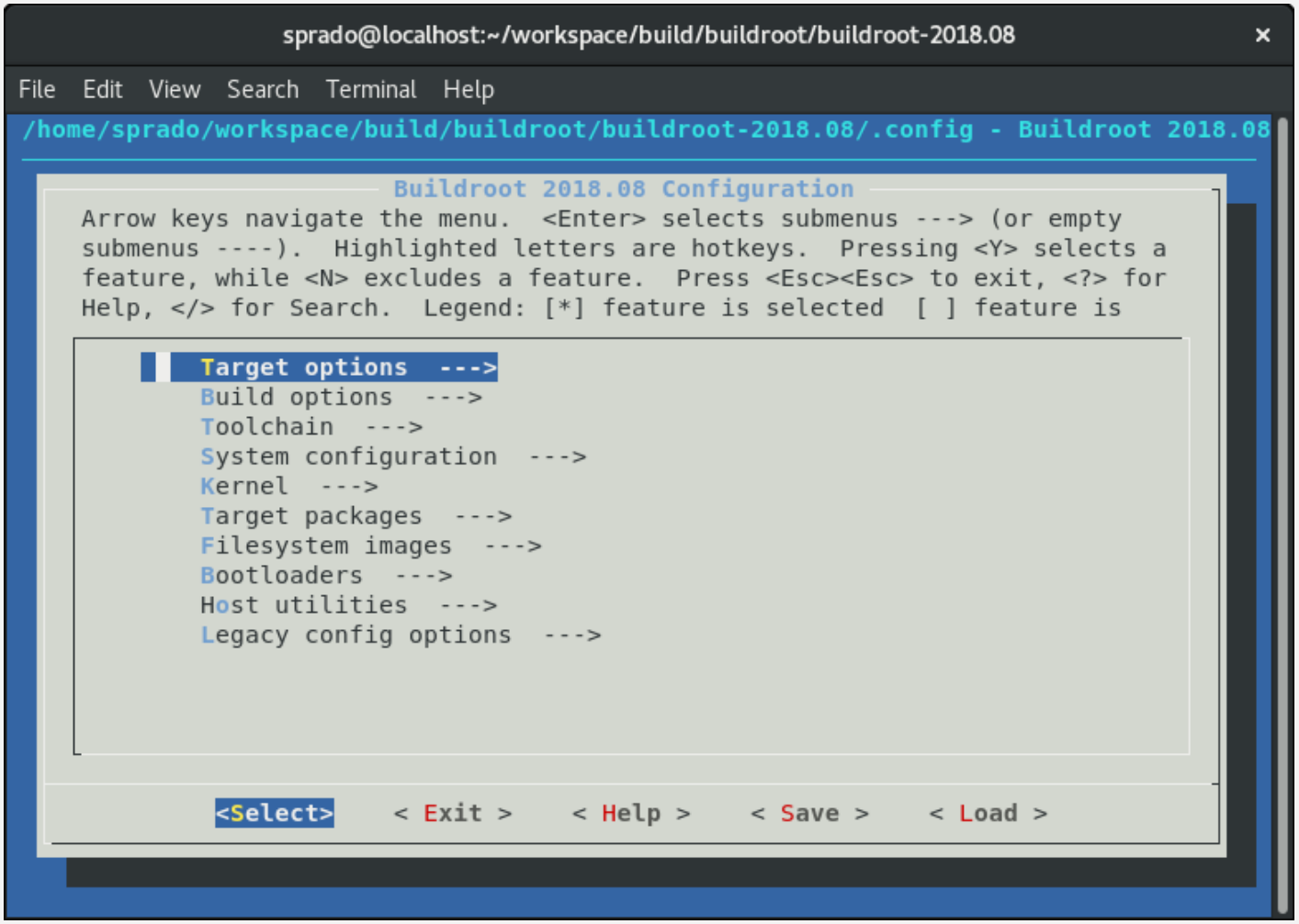
Fonte: http://free-electrons.com

CONFIGURANDO O BUILDROOT

- Permite configurar, dentre outras opções:
  - Arquitetura e modelo da CPU.
  - Toolchain.
  - Bootloader.
  - Kernel.
  - Bibliotecas e aplicações.
  - Tipos das imagens do rootfs (ext4, ubifs, etc).
- Para configurar:

```
$ make menuconfig
```

CONFIGURANDO O BUILDROOT (cont.)



COMPILANDO O BUILDROOT

- Configuração fica armazenada em um arquivo chamado `.config`.
- Para compilar:

```
$ make
```

- No final do processo de compilação, as imagens estarão disponíveis no diretório `output/images/`:

```
$ ls output/images/
imx6dl-colibri-ipe.dtb  rootfs.tar  u-boot.bin
rootfs.ext2             sdcard.img  u-boot.img
rootfs.ext4             SPL         zImage
```

Embedded Labworks

QtCon Brasil 2018

Laboratório 1: Gerando uma distribuição GNU/Linux com o Qt5

Embedded Labworks

QtCon Brasil 2018

Desenvolvendo aplicações com o Qt Creator

INTRODUÇÃO AO QT

- O Qt é um framework completo para o desenvolvimento de aplicações multiplataforma desktop e mobile.

  https://www.qt.io/

- Suporta inúmeras plataformas, incluindo os sistemas operacionais GNU/Linux, Windows, MacOS, Android e iOS e as arquiteturas x86, x86-64 e ARM.

- Diversas aplicações e produtos famosos utilizam o Qt, incluindo as smart TVs da LG e os softwares Autodesk Maya, Google Earth e VirtualBox.

  https://resources.qt.io/customer-stories-all

- É implementado em C++ mas possui bindings para outras linguagens, incluindo Python, Go, Rust, PHP e Java.

HISTÓRICO DO QT

- 1990: começou a ser escrito pela Trolltech.
- 1995: primeiro release público do Qt (licenças GPLv2 e comercial).
- 2008: Nokia compra a Trolltech.
- 2009: Lançado o Qt Creator e o Qt 4.5 (licença LGPLv2).
- 2011: Digia compra da Nokia a licença comercial e os direitos do Qt.
- 2014: Digia cria a subsidiária "The Qt Company".
- 2016: Lançamento do Qt 5.7, onde a maioria do módulos é relicenciado para LGPLv3.

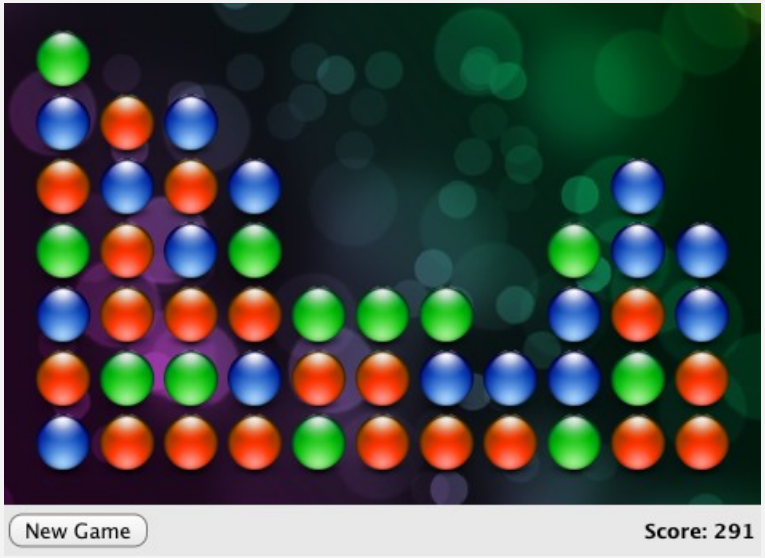
APIs

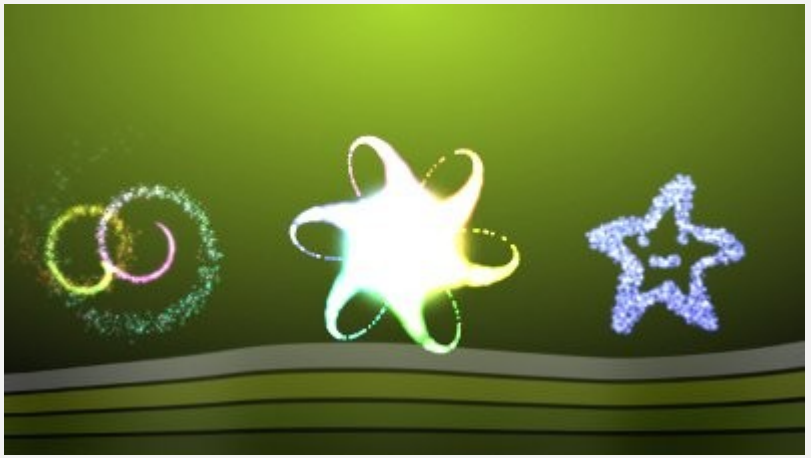
- O Qt possui uma API completa para o desenvolvimento de aplicações, incluindo bibliotecas para trabalhar com arquivos, threads, networking, banco de dados, multimedia, localização, gráficos, etc.
- Possui uma API baseada em widgets para o desenvolvimento de aplicações com interface gráfica.
- Através do módulo Qt Quick e da linguagem declarativa QML, possibilita o desenvolvimento de aplicações gráficas ricas e fluídas.

QT QUICK

- Na mudança do Qt4 para o Qt5, houve um foco grande no desenvolvimento de interfaces gráficas ricas através de um módulo chamado Qt Quick, baseado em três principais componentes:
  - Uma linguagem declarativa chamada QML.
  - Um interpretador Javascript.
  - Elementos para o desenvolvimento de uma interface gráfica rica, com bastante foco em animação e efeitos 3D.
- Apesar da grande quantidade de novos recursos, o Qt5 mantém compatibilidade com as versões anteriores do Qt.

EXEMPLOS QT QUICK





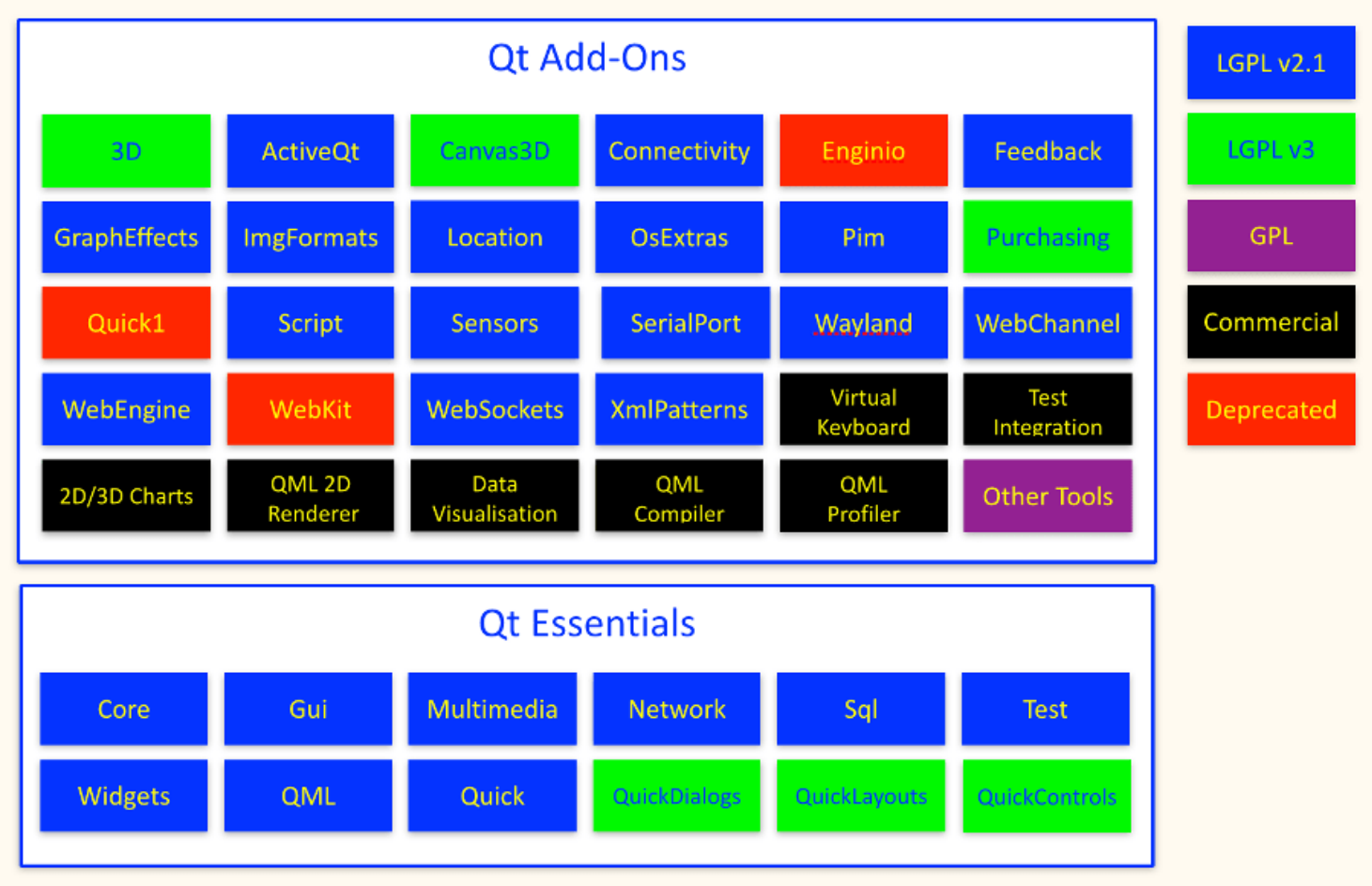
LICENÇAS

- Até a versão 5.3, os principais módulos do Qt eram licenciados com LGPLv2, uma licença bastante amigável do ponto de vista comercial.

- A partir do Qt 5.4, os principais módulos do Qt foram aos poucos relicenciados para LGPLv3, que pode ser uma barreira para produtos comerciais. Para estes casos, pode ser necessário adquirir uma licença comercial do Qt.

- Nas últimas versões do Qt, a maioria dos módulos tem licença LGPLv3.

LICENÇA (Qt v5.6)



Fonte: https://www.embeddeduse.com/2016/04/10/using-qt-5-6-and-later-under-lgpl/

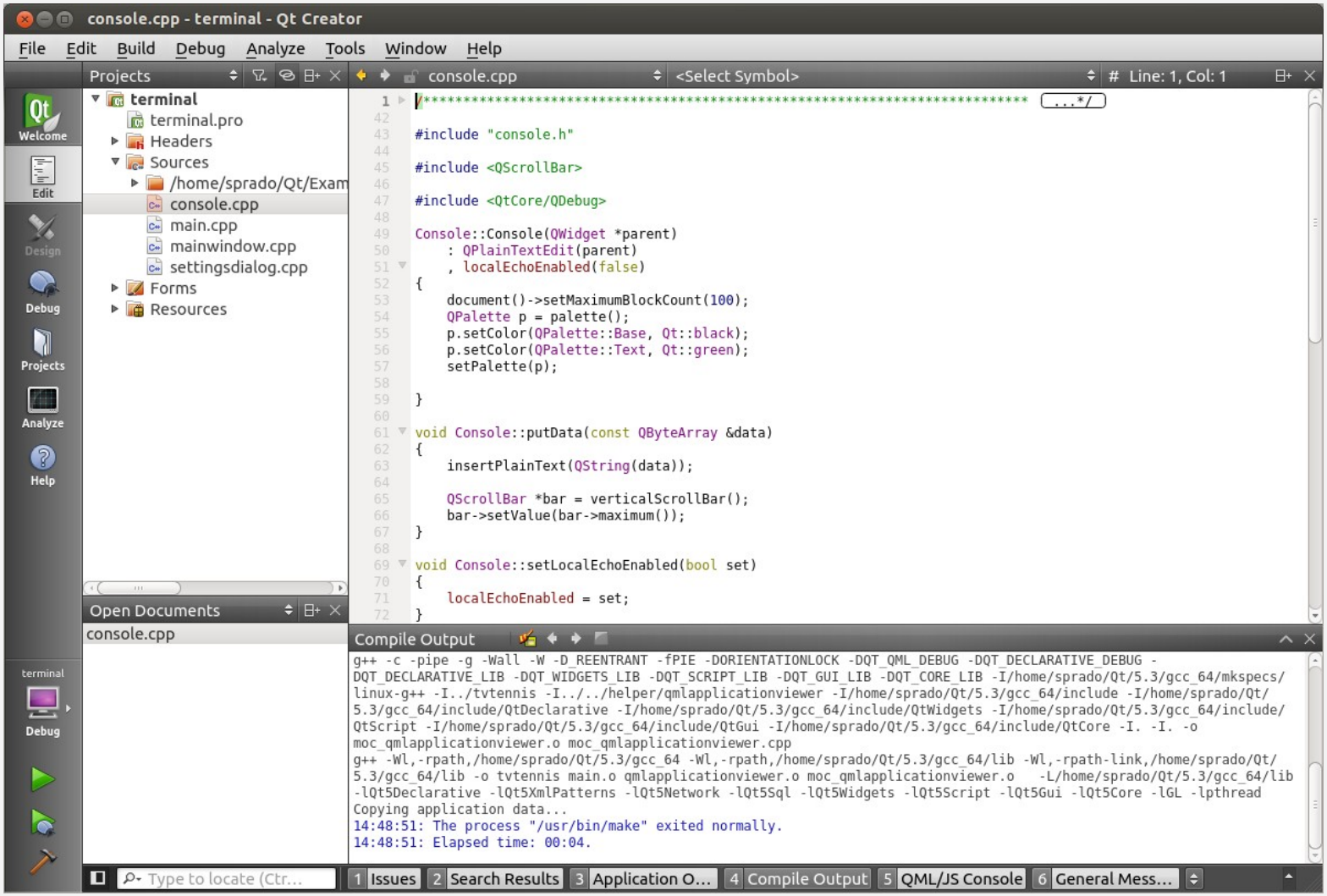
LICENÇA (Qt v5.7)

Fonte: https://www.embeddeduse.com/2016/04/10/using-qt-5-6-and-later-under-lgpl/

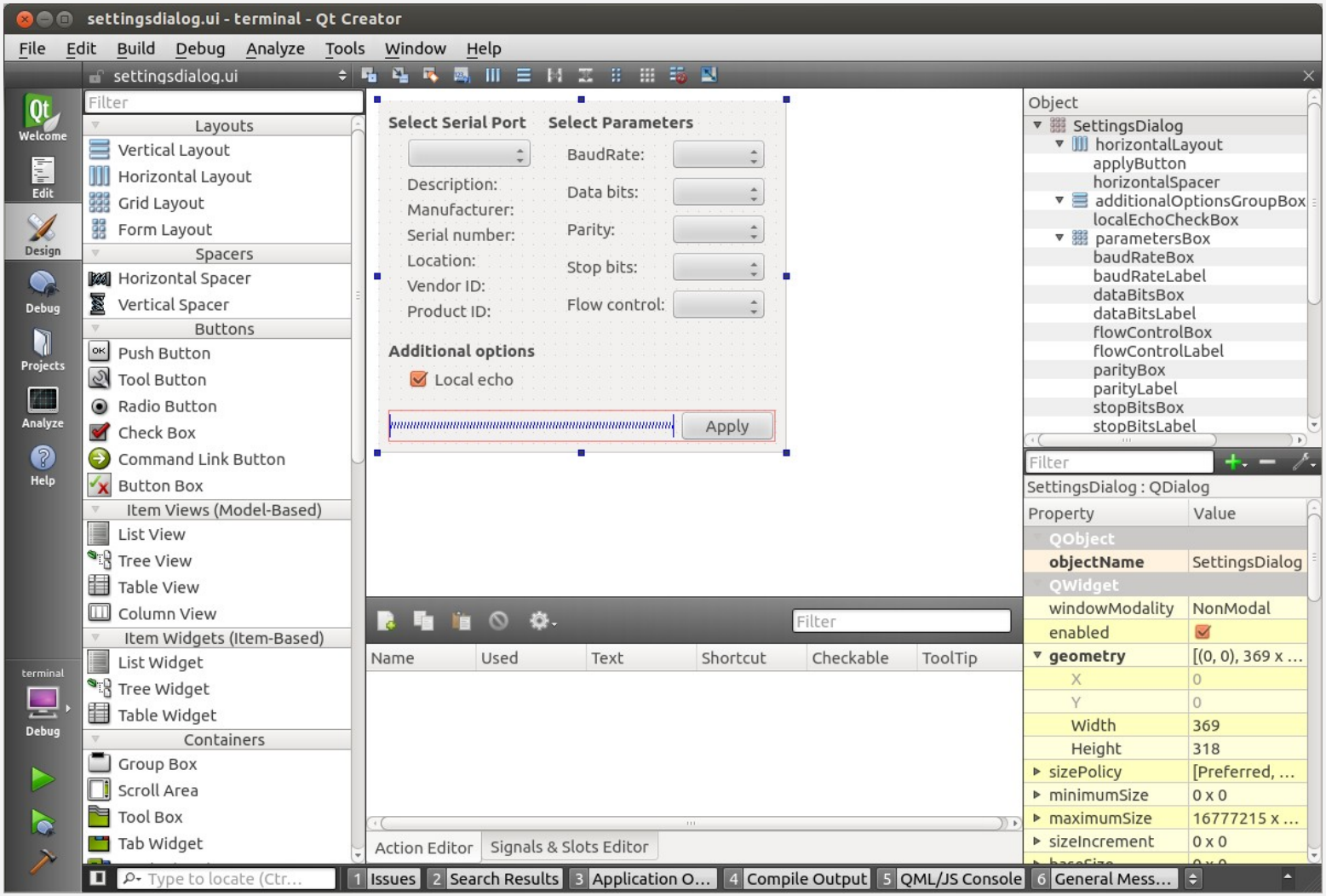
QT CREATOR

- Ambiente de desenvolvimento integrado (IDE) completo para o Qt.
  https://www.qt.io/download
- Disponível para Linux, Windows e MacOS.
- Suporta o desenvolvimento de aplicações em modo texto, aplicações gráficas através de widgets e aplicações multimídia ricas utilizando QML.
- Possibilita o desenvolvimento, testes e depuração remota em um dispositivo com Linux embarcado.

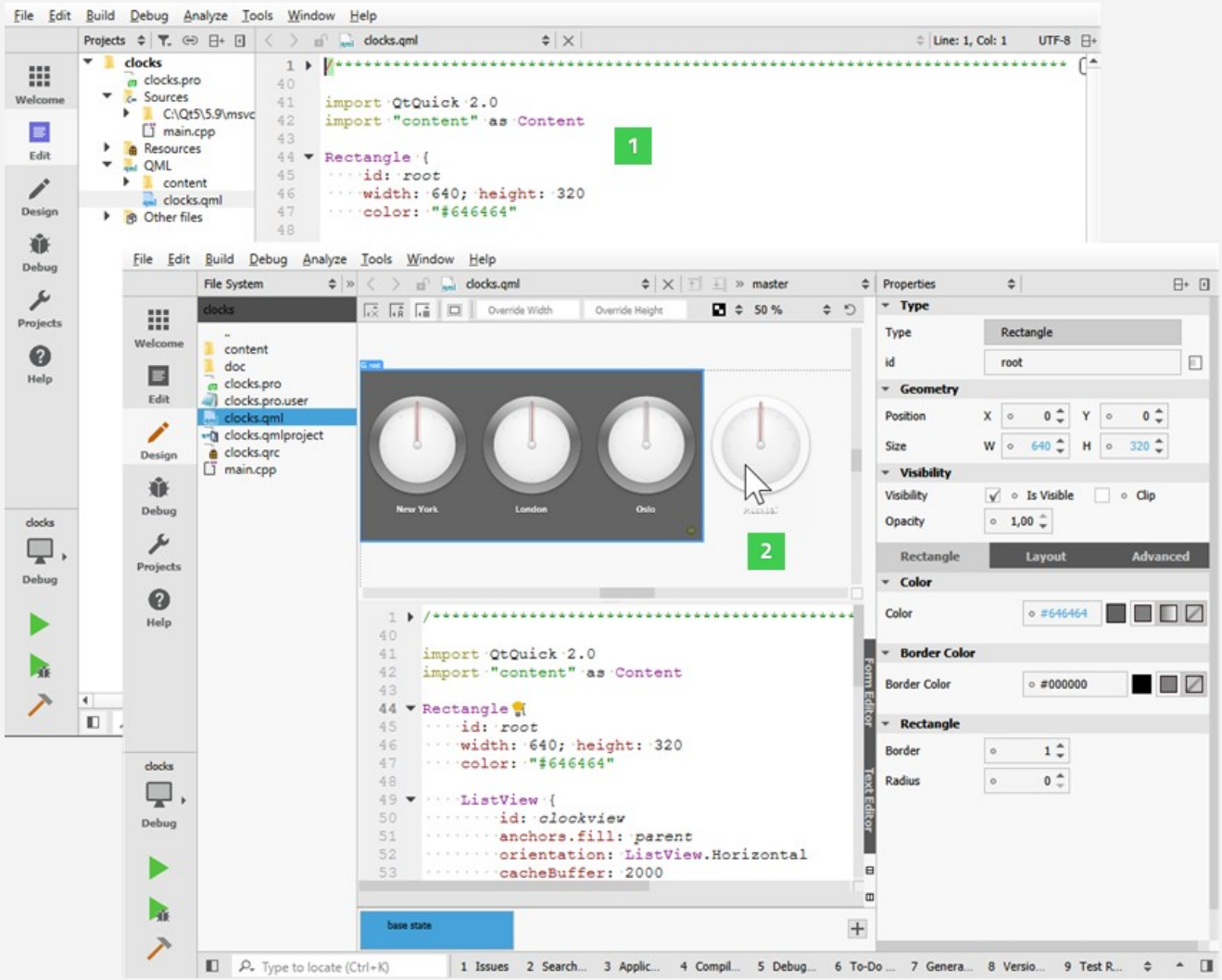
QT CREATOR (EDITOR)



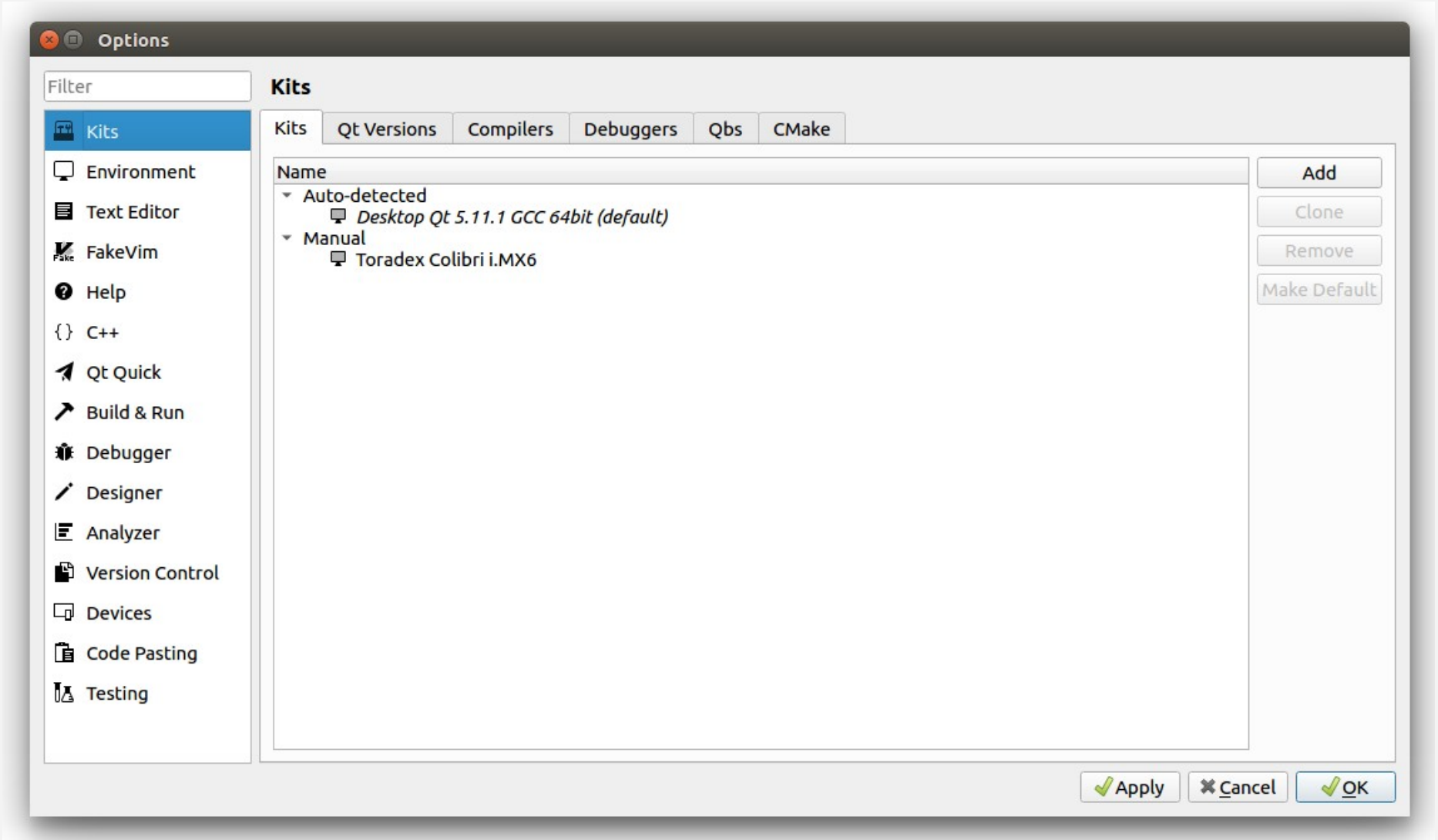
QT CREATOR (WIDGETS)



QT CREATOR (QML)



QT CREATOR (OPTIONS MENU)



Embedded Labworks

QtCon Brasil 2018

Laboratório 2: Configurando o Qt Creator

DÚVIDAS?

E-mail sergio.prado@e-labworks.com
Website https://e-labworks.com
Blog https://sergioprado.org
Twitter @sergioprado

Embedded **Labworks**

Por Sergio Prado. São Paulo, Novembro de 2018

QtCon BR

Aplicações Embarcadas com Qt5

Cleiton Bueno – B2Open Systems
cleiton.bueno@b2open.com

Direitos do material

Compartilhar — copiar e redistribuir o material em qualquer suporte ou formato.

Adaptar — remixar, transformar, e criar a partir do material para qualquer fim, mesmo que comercial.

O licenciante não pode revogar estes direitos desde que você respeite os termos da licença.
BY: Licença Creative Commons BY-SA 3.0

https://creativecommons.org/licenses/by-sa/3.0/br/

QtCon BR

B2Open Systems

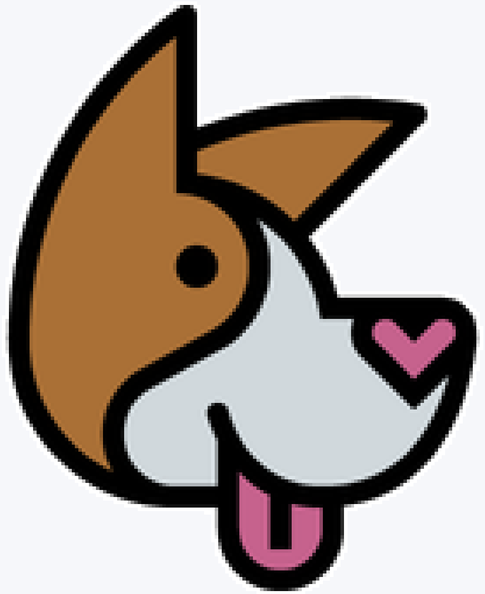
Trabalhamos com diversas soluções em FOSS realizando consultoria, treinamento, projetos alinhado ao cliente de diversos setores da industria entre estas soluções o toolkit Qt5.

@cleiton.bueno.7

@cleitonrbueno

@cleitonbueno

Fedora’s Tips



Quando ver este ícone em qualquer slide será abordado alguma dica ou macete solucionando algum problema ou evitando um, além de dicas que podem evitar horas e horas de pesquisa para soluções simples.

PS: Fedora é o nome da minha cachorra!

QtCon BR


Qt5 x ROI
QtCon
BR

Framework Qt5 justificando o ROI

1. Extenso numero de bibliotecas e modular.

Framework Qt5 justificando o ROI

1. Extenso numero de bibliotecas e modular.
2. Aplicações Multiplataformas.

Framework Qt5 justificando o ROI

1. Extenso numero de bibliotecas e modular.
2. Aplicações Multiplataformas.
3. **IDE QtCreator:** IDE completa para desenvolver, compilar, depurar, profilar e configuração de diversos Kits para diferentes plataformas.

Framework Qt5 justificando o ROI

1. Extenso numero de bibliotecas e modular.
2. Aplicações Multiplataformas.
3. **IDE QtCreator:** IDE completa para desenvolver, compilar, depurar, profilar e configuração de diversos Kits para diferentes plataformas.
4. **Qt Linguist:** Internacionalização com Qt.

Framework Qt5 justificando o ROI

1. Extenso numero de bibliotecas e modular.
2. Aplicações Multiplataformas.
3. **IDE QtCreator:** IDE completa para desenvolver, compilar, depurar, profilar e configuração de diversos Kits para diferentes plataformas.
4. **Qt Linguist:** Internacionalização com Qt.
5. QPA para Linux Embarcado: eglfs, linuxfb, wayland, XBC, …

Framework Qt5 justificando o ROI

1. Extenso numero de bibliotecas e modular.
2. Aplicações Multiplataformas.
3. **IDE QtCreator:** IDE completa para desenvolver, compilar, depurar, profilar e configuração de diversos Kits para diferentes plataformas.
4. **Qt Linguist:** Internacionalização com Qt.
5. QPA para Linux Embarcado: eglfs, linuxfb, wayland, XBC, …
6. **Qt VirtualKeyboard:** Teclado virtual com suporte a mais de 35 idiomas.

Framework Qt5 justificando o ROI

1. Extenso numero de bibliotecas e modular.
2. Aplicações Multiplataformas.
3. **IDE QtCreator:** IDE completa para desenvolver, compilar, depurar, profilar e configuração de diversos Kits para diferentes plataformas.
4. **Qt Linguist:** Internacionalização com Qt.
5. QPA para Linux Embarcado: eglfs, linuxfb, wayland, XBC, …
6. **Qt VirtualKeyboard:** Teclado virtual com suporte a mais de 35 idiomas.
7. **Qt OTA:** Over-The-Air baseado no OSTree.

Framework Qt5 justificando o ROI

1. Extenso número de bibliotecas e modular.
2. Aplicações Multiplataformas.
3. **IDE QtCreator:** IDE completa para desenvolver, compilar, depurar, profilar e configuração de diversos Kits para diferentes plataformas.
4. **Qt Linguist:** Internacionalização com Qt.
5. QPA para Linux Embarcado: eglfs, linuxfb, wayland, XBC, …
6. **Qt VirtualKeyboard:** Teclado virtual com suporte a mais de 35 idiomas.
7. **Qt OTA:** Over-The-Air baseado no OSTree.
8. **Qt Device Creation:** SO Linux Customizado e otimizado para Sistemas Embarcado.

Toradex
Embedded. Computing.

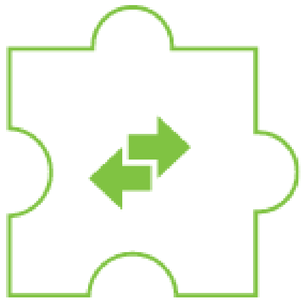

Framework
Qt5
QtCon
BR

Cross-Platform Class Library

One Technology for All Platforms

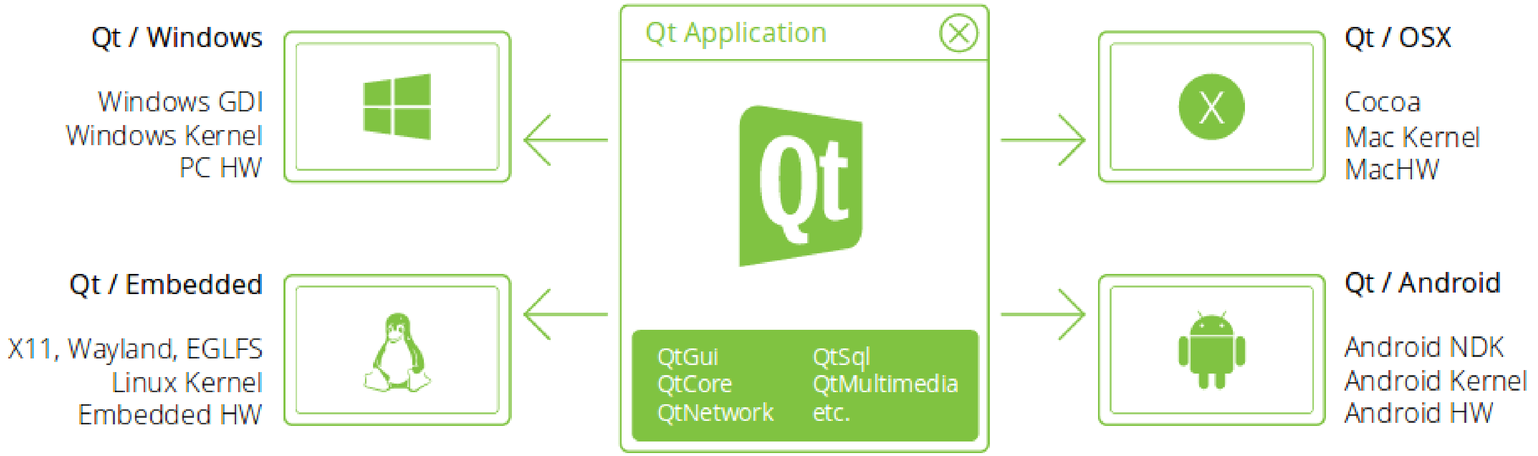
Integrated Development Tools

Shorter Time-to-Market



Cross-Platform IDE, Qt Creator

Productive development environment


Qt / Windows
Windows GDI
Windows Kernel
PC HW
Qt / Embedded
X11, Wayland, EGLFS
Linux Kernel
Embedded HW
Qt Application
QtGui
QtCore
QtNetwork
QtSql
QtMultimedia
etc.
Qt / OSX
Cocoa
Mac Kernel
MacHW
Qt / Android
Android NDK
Android Kernel
Android HW

Módulos

Módulos contêm bibliotecas, plugins e documentação. Alguns módulos são comuns para todas plataformas, outros, são de proposito especifico e de uma plataforma alvo.

Qt Essentials

Disponíveis para todas plataformas QtCore, QtGui, QtMultimedia, QtNetwork, QtQML, QtQuick, QtSQL, QtTest, QtWidgets.

Qt Add-ons

Módulos de proposito especifico nem sempre disponível para todas plataformas Qt3D, QtBluetooth, QtConcurrent, QtD-BUS, QtGamePad, QtLocation, QtNFC, QtPositioning, QtPrint, QtQuickControls1/2, QtSensors, QtSerialBus, QtSerialPort e etc...

Destaque para Embarcados

QtSerialPort
QtMqtt
QtSerialBus → QCanBus e QModbus*
QtKnx
QCryptographicHash
QDateTime
QRegularExpression
QSettings
QCommandLineParser
QFileSystemWatcher
QJson*
IPC
Qt Sql
Qt Networking

#1

Extensões

.qrc – Qt Resource Collection (armazenar icones, arquivos de traduções, imagens e etc)
.cpp – Código-fonte C++
.h – Cabecalho do Código-fonte C++
.ui – User Interface
.qml – Qt Modeling Language (linguagem declarativa, fluida, semelhante com JSON para descrever GUI)
.js – Contendo rotinas e declarações JavaScript
.pro – Projeto, descrição do projeto e dependências, utilizado pelo qmake
.pri – Includes a serem estendidos no .pro

Quem utiliza?


QtQuick
(QML)
QtCon
BR

Qt Quick

Utiliza uma linguagem declarativa conhecida como QML, e imperativa com JavaScript, integração runtime com Qt, API C++ para integração e suporte da IDE Qt Creator para linguagem QML

Qt Quick

Fácil prototipação e pode-se visualizar a GUI sem uma linha de C++.

Qt Quick1 Preview: qmlviewer

Qt Quick2 Preview: qmlscene

Qt Quick

Qt Quick2 Preview: qmlscene

A ferramenta qmlscene carrega e exibe documentos QML antes mesmo da conclusão do aplicativo. Esse utilitário também fornece os seguintes recursos adicionais que são úteis ao desenvolver aplicativos QML.

Qt Quick

Inicialmente em QML vamos falar de Itens e Propriedades.

Qt Quick

Inicialmente em QML vamos falar de Itens e Propriedades

Itens: É o item básico de todo componente visual, onde se tem posições, cores, ancoras entre diversos outras opções.

Qt Quick

Inicialmente em QML vamos falar de Itens e Propriedades

Itens: É o item básico de todo componente visual, onde se tem posições, cores, ancoras entre diversos outras opções.

Propriedades: É uma relação chave-valor, onde se configura/define valores no caso, as posições, cores, valores dentro de um Item.

Primeiro QML

```
Window {
    visible: true
    width: 400
    height: 150
    title: qsTr("QtCon Brasil")

    Rectangle {
        id: rect1
        width: 200; height: 100;
        x: 50
        y: 10
        color: "green"
    }
}
```

Primeiro QML

```
Window {
    visible: true
    width: 400
    height: 150
    title: qsTr("QtCon Brasil")

    Rectangle {
        id: rect1
        width: 200; height: 100;
        x: 50
        y: 10
        color: "green"
    }
}
```

Itens

Properties

Primeiro QML

```
Window {
    visible: true
    width: 400
    height: 150
    title: qsTr("QtCon Brasil")

    Rectangle {
        id: rect1
        width: 200; height: 100;
        x: 50
        y: 10
        color: "green"
    }
}
```

Tools > External > Qt Quick > Qt Quick 2 Preview
ou
/home/qtcon/Qt/5.11.1/gcc_64/bin/qmlscene

Erro deploy/run remoto

qt.qpa.plugin: Could not find the Qt platform plugin ""

This application failed to start because no Qt platform plugin could be initialized. Reinstalling the application may fix this problem.

Available platform plugins are: eglfs, minimal, minimalegl, offscreen.

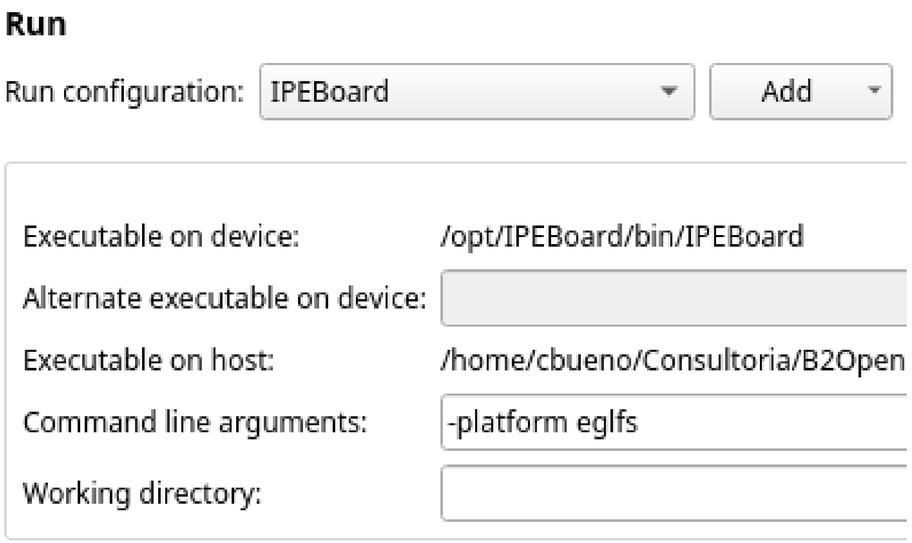
12:48:43: Process killed by signal

#2

Erro deploy/run remoto

IDE Qt Creator > Projects(Ctrl+5) > Selecionar Kit(ToradexQtCon) > Run

Procurar por Run e adicionar a plataforma em uso em "**Command line arguments**"



#2

Cores

Tipos de especificação de cores

SVG Names: “yellow”, “red”, “green”, “gold”

HTML Style Color: “#0000FF”

Qt built-in: Qt.rgba(0, 0.75, 0, 1)

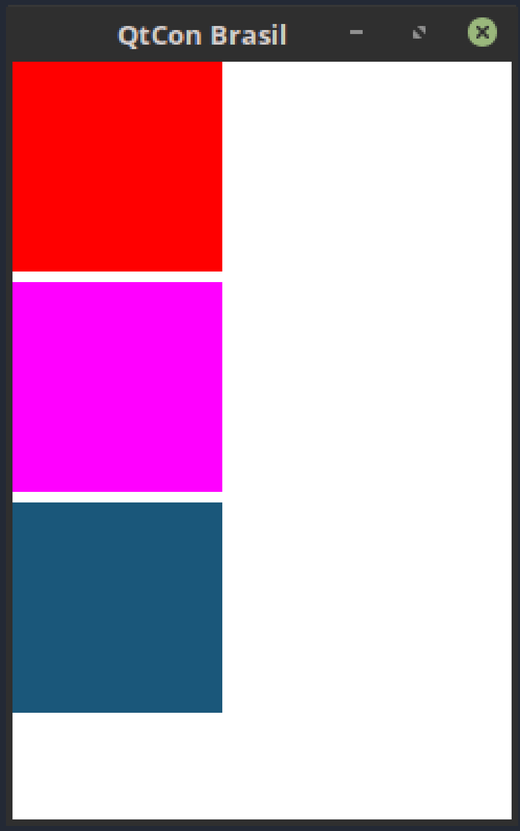
Posicionamento

Posicionamento em QML é falar de Column, Grid, Row e estender com Anchors.

Posicionamento

Column

```
Column {
    spacing: 5
    Rectangle { width: 100; height: 100; color: "red" }
    Rectangle { width: 100; height: 100; color: "#FF00FF" }
    Rectangle { width: 100; height: 100; color: Qt.rgba(0.1, 0.34, 0.48, 1) }
}
```



Posicionamento

Grid

```
Grid {
    spacing: 5
    columns: 2
    //rows: 2
    Rectangle { width: 100; height: 100; color: "red" }
    Rectangle { width: 100; height: 100; color: "#FF00FF" }
    Rectangle { width: 100; height: 100; color: Qt.rgba(0.1, 0.34, 0.48, 1) }
}
```

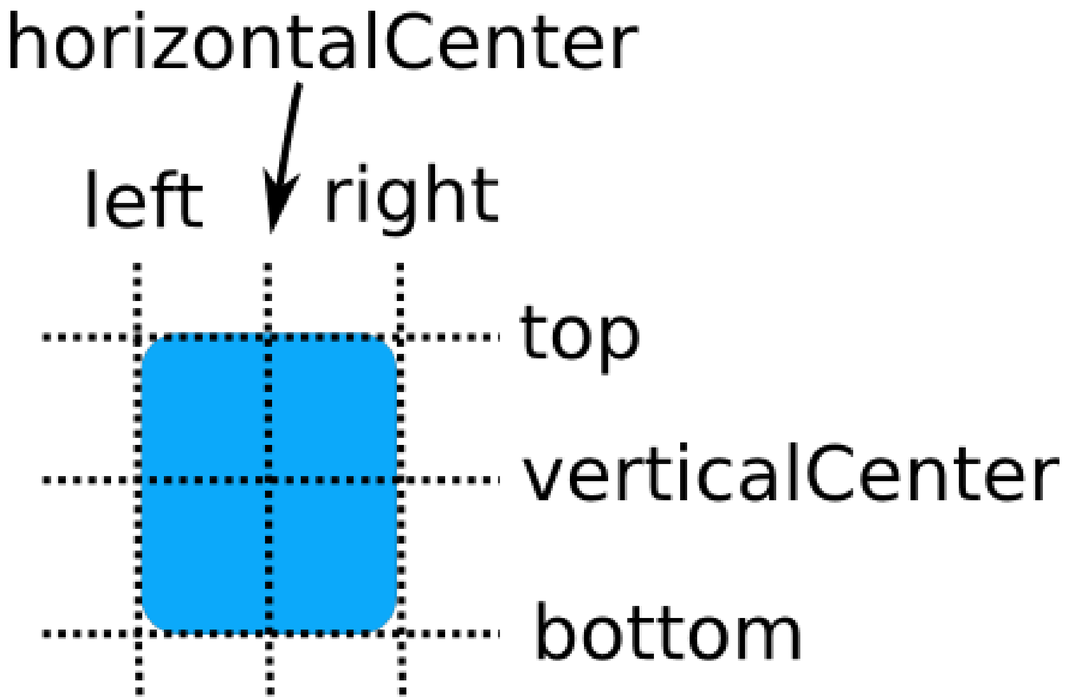
Posicionamento

Row

```
Row {
    spacing: 5
    Rectangle { width: 100; height: 100; color: "red" }
    Rectangle { width: 100; height: 100; color: "#FF00FF" }
    Rectangle { width: 100; height: 100; color: Qt.rgba(0.1, 0.34, 0.48, 1) }
}
```

Posicionamento

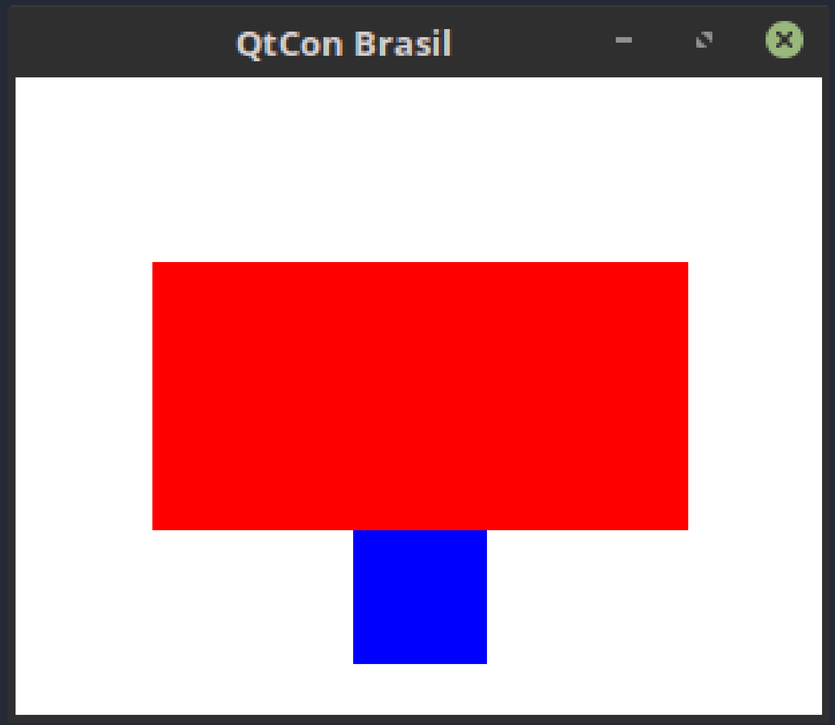
Anchors



Posicionamento

Anchors

```
Rectangle {
    id: rect1
    width: 200; height: 100;
    anchors.centerIn: parent; color: "red"
}

Rectangle {
    id: rect2
    width: 50; height: 50
    color: "blue"
    anchors.horizontalCenter: rect1.horizontalCenter
    anchors.top: rect1.bottom
}
```



Imagens

```
ApplicationWindow {
    visible: true
    width: 640
    height: 480
    title: qsTr("Lab01 - QtCon")

    Rectangle {
        width: parent.width/2
        height: parent.height/2
        color: "gold"
        anchors.centerIn: parent

        Image {
            width: parent.width*0.9
            height: parent.height*0.9
            anchors.centerIn: parent
            source: "qrc:///RES/IMAGES/bg-dark-green.jpg"
        }
    }

}
```

Imagem não suportada

qrc:/main.qml:16:9: QML Image: Error decoding: qrc:///RES/IMAGES/bg-dark-green.jpg: Unsupported image format

#3

Imagem não suportada

Listando os formatos de imagens suportadas para carregar e salvar imagens:

```
QImageReader::supportedImageFormats();
QImageWriter::supportedImageFormats();
```

#3

Imagem não suportada

Host

Formatos para carregar imagens: ("bmp", "cur", "gif", "icns", "ico", "**jpeg**", "**jpg**", "pbm", "pgm", "**png**", "ppm", "svg", "svgz", "tga", "tif", "tiff", "wbmp", "webp", "xbm", "xpm")

Formatos para salvar imagens : ("bmp", "cur", "icns", "ico", "**jpeg**", "**jpg**", "pbm", "pgm", "**png**", "ppm", "tif", "tiff", "wbmp", "webp", "xbm", "xpm")

#3

Imagem não suportada

Target

Formatos para carregar imagens: ("bmp", "cur", "icns", "ico", "pbm", "pgm", "**png**", "ppm", "svg", "svgz", "tga", "tif", "tiff", "wbmp", "webp", "xbm", "xpm")

Formatos para salvar imagens : ("bmp", "cur", "icns", "ico", "pbm", "pgm", "**png**", "ppm", "tif", "tiff", "wbmp", "webp", "xbm", "xpm")

Solução: Usar o formato png ou algum outros da lista de suportados!

#3

Texto

```
ApplicationWindow {
    visible: true
    width: 400
    height: 150
    title: qsTr("QtCon Brasil")

    Rectangle {
        id: rect1
        anchors.fill: parent
        color: "green"

        Text {
            color: "white"
            anchors.centerIn: parent
            font.pointSize: 32
            text: "QtCon 2018"
        }
    }
}
```

Eventos do Teclado

Captura os eventos do teclado, porém, deverá setar ‘**focus:** true’ no item a utilizar Keys{}

```
Rectangle {
    id: rect1
    anchors.fill: parent
    color: "green"
    focus: true

    Text { ...}

    Keys.onPressed: {
        if (event.key === Qt.Key_Left) {
            console.log("<");
        }
        else if (event.key === Qt.Key_Right) {
            console.log(">")
        }
    }
}
```

Lembre-se deste Keys no Laboratório!

MouseArea

Capturar toque na tela touchscreen, utilizar MouseArea{}

```
// Button Power Off
Rectangle {
    width: 64
    height: 64
    color: Qt.darker("red")
    anchors.top: parent.top
    anchors.right: parent.right
    anchors.margins: 10
    radius: width/2

    Image { ...}

    MouseArea {
        anchors.fill: parent
        onPressed: {
            parent.scale = 0.95
        }
        onReleased: {
            parent.scale = 1.0
        }
    }
}
```

Framework Qt5 - C++

QtCon BR

C++

```cpp
class Log {

public:
    explicit Log();
    explicit Log(char const *file);
    ~Log();

    void setPathLog(char const *file);

private:
    size_t m_size_log;
    std::unique_ptr<std::ofstream> m_out_stream;

};
```

C++ / Qt

```cpp
#include <QObject>

class Tasks : public QObject
{
    Q_OBJECT
public:
    explicit Tasks(QObject *parent = nullptr);
    bool readDmesg(void);

    Q_INVOKABLE bool detectTouchScreen() const;

private:
    bool m_devTouch;

signals:
    void finishReadDmesg();

public slots:
    void processDmesgData(const QString *data);
};
```

QFile

Classe que fornece uma interface amigável e completa para se manipular arquivos.

open(), close(), seek(), write(), read(), rename(), remove(), ...

Estendendo recursos com:

QByteArray: Array de bytes, usa CoW[copy-on-write)

QTextStream: Dados Formatados

QDataStream: Dados Binarios

QString: Poderosa classe para manipulação de strings

QFile

```
QFile file("/tmp/dataIn.txt");
if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
    qCritical() << "Falha ao abrir dados de temperatura.";
    return;
}
QTextStream readFile(&file);
while (!readFile.atEnd()) {
    QString line = readFile.readLine();
    qDebug() << "Temperatura: " << line;
}
```

Saída:

```
Temperatura:  "29.5"
Temperatura:  "30.2"
Temperatura:  "27.6"
Temperatura:  "28.8"
```

QDebug

A classe QDebug fornece um fluxo de saída para informações de depuração.

O QDebug é usado sempre que o desenvolvedor precisa gravar informações de depuração ou de rastreamento em um dispositivo, arquivo, string ou console.

QDebug

```
1. qDebug("Mensagem1");
2. qDebug() << "Debug:" << endl;
3. qInfo() << "Info:";
4. qWarning() << "Atencao:";
5. qCritical() << "Erro Critico:";
6. qFatal("Erro fatal, encerrando
   aplicação");
```

QTimer

Classe de um temporizador, para eventos repetitivos ou um único disparo com atraso, por exemplo.

```
1. QTimer *timer = new QTimer(this);
2. connect(timer, SIGNAL(timeout()), this,
       SLOT(update()));
3. timer->start(1000);
4.
5. QTimer::singleShot(200, this,
   SLOT(updateCaption()));
```


Sinais e
Slots
QtCon
BR

Sinais e Slots

Se QObject é o coração do Qt Object Model, podemos dizer que sinais e slots são as artérias!

Sinais e Slots

Se QObject é o coração do Qt Object Model, podemos dizer que sinais e slots são as artérias!

Principal mecanismo de comunicação entre os objetos criados em C++ no Qt5, necessário mencionar **Q_OBJECT** na classe, para o MOC saber que esta classe utiliza Sinais e Slots.

Sinais e Slots

Se QObject é o coração do Qt Object Model, podemos dizer que sinais e slots são as artérias!

Principal mecanismo de comunicação entre os objetos criados em C++ no Qt5, necessário mencionar **Q_OBJECT** na classe, para o MOC saber que esta classe utiliza Sinais e Slots.

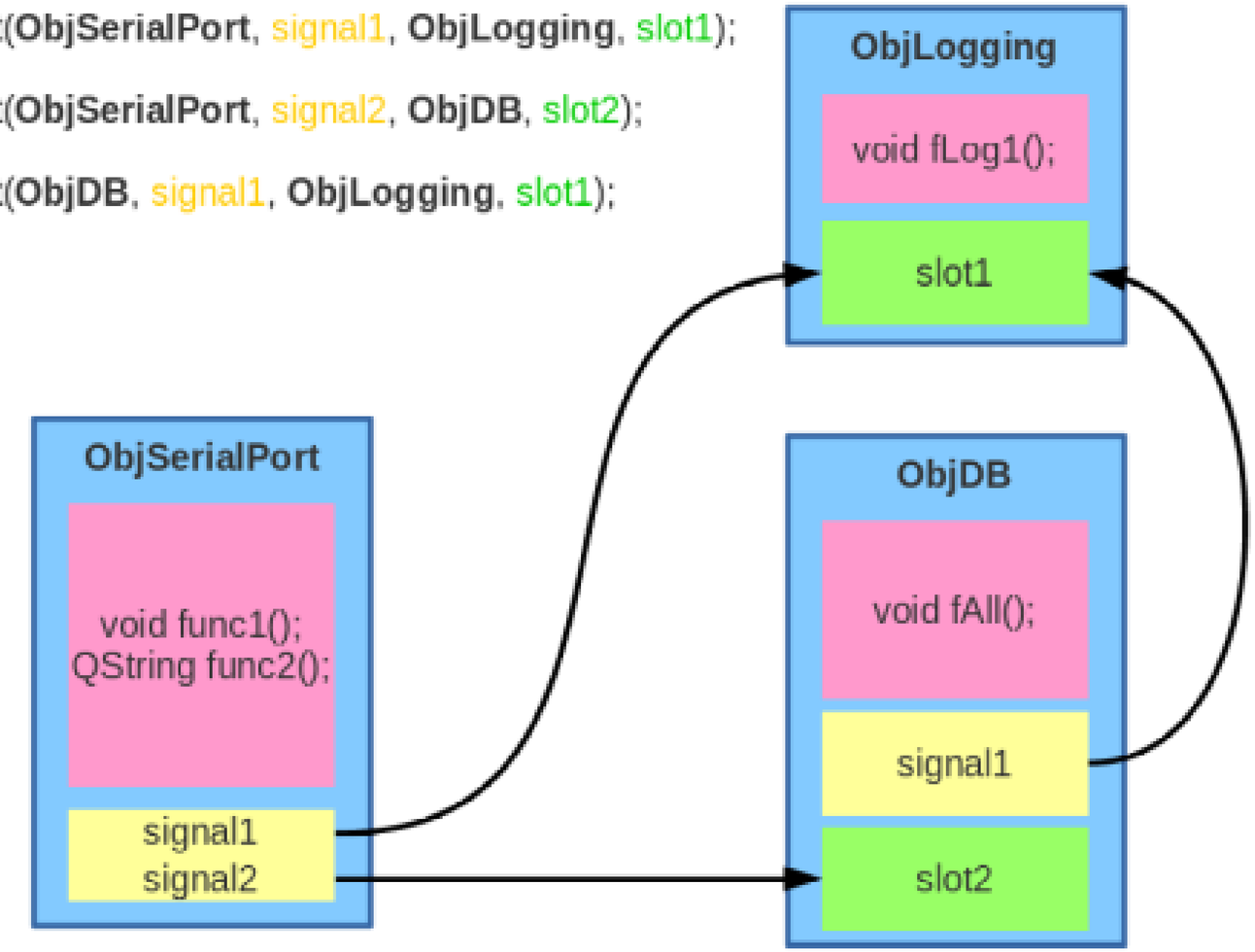
A classes que utilizam sinais e slots, também devem derivar (direta ou indiretamente) do QObject.

Sinais e Slots

connect(**ObjSerialPort**, signal1, **ObjLogging**, slot1);

connect(**ObjSerialPort**, signal2, **ObjDB**, slot2);

connect(**ObjDB**, signal1, **ObjLogging**, slot1);



Sinais e Slots

Estilo Qt4

```
connect(&objeto1, SIGNAL(sigObj1()),
        objeto2, SLOT(slotObj2(QString))
);
```

Estilo Qt5

```
connect(&objeto1, &Class1::sigObj1,
        objeto2, &Class2::slotObj2
);
```

Sinais e Slots

Estilo Qt5 com C++11 Lambda Expressions

```
connect(&objeto1, &Class1::sigObj1,
        [=](const QString &s) {
          // Rotinas aqui!
          qDebug() << “Atualizado!”;
        }
);
```

Sinais e Slots

Evitando conflitos e problemas com bibliotecas de terceiro, não usando **signals**, **slot** e **emit**.

Solução:

Adicionar no .pro:

CONFIG += no_keywords

Utilizar:

signals → Q_SIGNAL() || Q_SIGNALS()
slots → Q_SLOT() || Q_SLOTS()
emit → Q_EMIT()

#4

Integrando QML e C++

QtCon BR

A magica

Quem faz todo trabalho duro para que esta funcionalidade pareça magica é o **Qt Meta-Object System** em especial neste curso o **QObject** e a macro Q_OBJECT.

A magica

Quem faz todo trabalho duro para que esta funcionalidade pareça magica é o **Qt Meta-Object System** em especial neste curso o **QObject** e a macro Q_OBJECT.

- **Q_PROPERTY**
- **Q_INVOKABLE**
- **Signals/Slots**

Q_PROPERTY

Recurso muito útil para configurar uma propriedade(set), ler o valor de um propriedade(get) ou notificar alguma alteração(signal).

Um protótipo comum usando **Q_PROPERTY**.

```
Q_PROPERTY(QString msg READ msg WRITE setMsg NOTIFY msgChanged)
```

Q_PROPERTY

```
class Acelerometro : public QObject
{
    Q_OBJECT
    Q_PROPERTY(int eixoX READ eixoX WRITE setEixoX NOTIFY changedEixoX)

public:
    explicit Acelerometro(QObject *parent = nullptr);
    ~Acelerometro();

    void setEixoX(const int value);
    int eixoX() const;

private:

    int m_value_accel_x;

signals:
    void changedEixoX();

public slots:

};
```

Q_INVOKABLE

Métodos que podem ser chamados direto do QML, e que podem ou não retornar algum dado, o mesmo vale para slots.

Um protótipo comum usando **Q_INVOKABLE**.

```
Q_INVOKABLE void registrarFalha(QString falha)
```

Q_INVOKABLE

```
class Led : public QObject
{
    Q_OBJECT
    Q_PROPERTY(int led  READ led  WRITE setLed  NOTIFY changedLed)

public:
    explicit Led(QObject *parent = nullptr);
    ~Led();

    void setLed(const int led);
    int led() const;

    Q_INVOKABLE void on();
    Q_INVOKABLE void off();

private:
    int m_led;

signals:
    void changedLed();

public slots:
};
```

QML

main.qml

```
import QtQuick 2.9
import QtQuick.Controls 2.3
import b2open.qt 1.0


ApplicationWindow {
    id: root
    visible: true
    width: 640
    height: 480

    Led { id: led101; led: Led.Led101; modo: Led.HeartBeat; }
    Led { id: led103; led: Led.Led103; modo: Led.Timer;     }
    Led { id: led133; led: Led.Led133;
```

C++

main.cpp

```
#include <QDebug>
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "led.h"

int main(int argc, char *argv[])
{
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QGuiApplication app(argc, argv);

    qmlRegisterType<Toradex::Led>("b2open.qt", 1, 0, "Led");

    QQmlApplicationEngine engine;
    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
    if (engine.rootObjects().isEmpty())
        return -1;

    return app.exec();
}
```

led.h

```
#include <QObject>

namespace Toradex {

class Led : public QObject
{
    Q_OBJECT
    Q_PROPERTY(int led                WRITE setLed   NOTIFY changedLed)
    Q_PROPERTY(int modo               WRITE setModo  NOTIFY changedModo)
    Q_PROPERTY(bool ligar  READ ligar WRITE setLigar NOTIFY changedLigar)

public:
    explicit Led(QObject *parent = nullptr);
    ~Led();

    void setLed(const int led);
    void setModo(const int modo);
    void setLigar(bool flag);
    bool ligar() const;
```

QtCon BR

QML

main.qml

```
import QtQuick 2.9
import QtQuick.Controls 2.3
import b2open.qt 1.0


ApplicationWindow {
    id: root
    visible: true
    width: 640
    height: 480

    Led { id: led101; led: Led.Led101; modo: Led.HeartBeat; }
    Led { id: led103; led: Led.Led103; modo: Led.Timer;     }
    Led { id: led133; led: Led.Led133;
```

C++

main.cpp

```
#include <QDebug>
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "led.h"

int main(int argc, char *argv[])
{
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QGuiApplication app(argc, argv);

    qmlRegisterType<Toradex::Led>("b2open.qt", 1, 0, "Led");

    QQmlApplicationEngine engine;
    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
    if (engine.rootObjects().isEmpty())
        return -1;

    return app.exec();
}
```

led.h

```
#include <QObject>

namespace Toradex {

class Led : public QObject
{
    Q_OBJECT
    Q_PROPERTY(int led                WRITE setLed   NOTIFY changedLed)
    Q_PROPERTY(int modo               WRITE setModo  NOTIFY changedModo)
    Q_PROPERTY(bool ligar  READ ligar WRITE setLigar NOTIFY changedLigar)

public:
    explicit Led(QObject *parent = nullptr);
    ~Led();

    void setLed(const int led);
    void setModo(const int modo);
    void setLigar(bool flag);
    bool ligar() const;
```

QtCon BR

QML

main.qml

```
import QtQuick 2.9
import QtQuick.Controls 2.3
import b2open.qt 1.0


ApplicationWindow {
    id: root
    visible: true
    width: 640
    height: 480

    Led { id: led101; led: Led.Led101; modo: Led.HeartBeat; }
    Led { id: led103; led: Led.Led103; modo: Led.Timer;     }
    Led { id: led133; led: Led.Led133;
```

C++

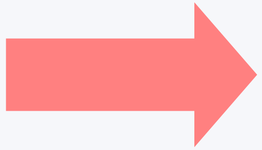
main.cpp

```
#include <QDebug>
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "led.h"

int main(int argc, char *argv[])
{
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QGuiApplication app(argc, argv);

    qmlRegisterType<Toradex::Led>("b2open.qt", 1, 0, "Led");

    QQmlApplicationEngine engine;
    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
    if (engine.rootObjects().isEmpty())
        return -1;

    return app.exec();
}
```

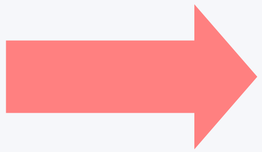
led.h

```
#include <QObject>

namespace Toradex {

class Led : public QObject
{
    Q_OBJECT
    Q_PROPERTY(int led                WRITE setLed   NOTIFY changedLed)
    Q_PROPERTY(int modo               WRITE setModo  NOTIFY changedModo)
    Q_PROPERTY(bool ligar  READ ligar WRITE setLigar NOTIFY changedLigar)

public:
    explicit Led(QObject *parent = nullptr);
    ~Led();

    void setLed(const int led);
    void setModo(const int modo);
    void setLigar(bool flag);
    bool ligar() const;
```

QtCon BR

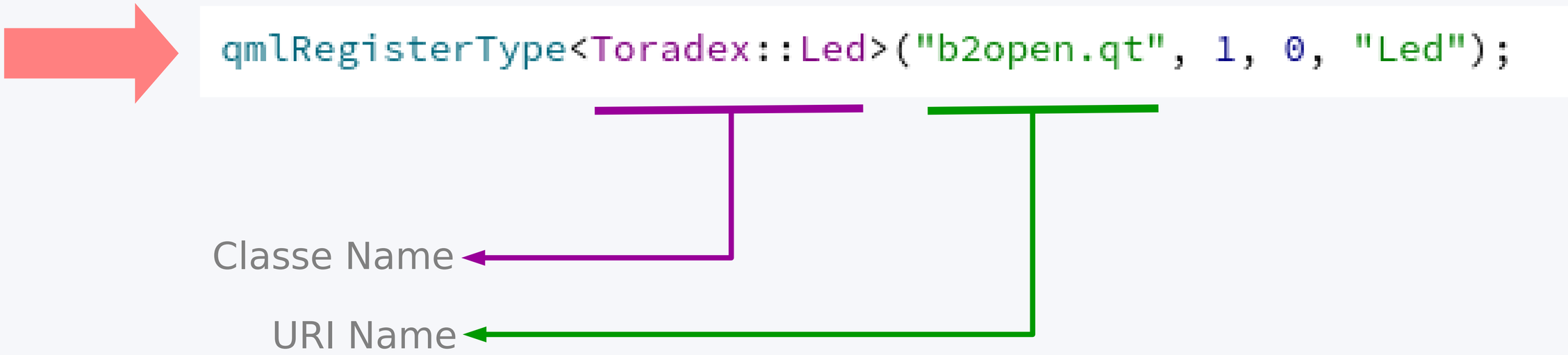


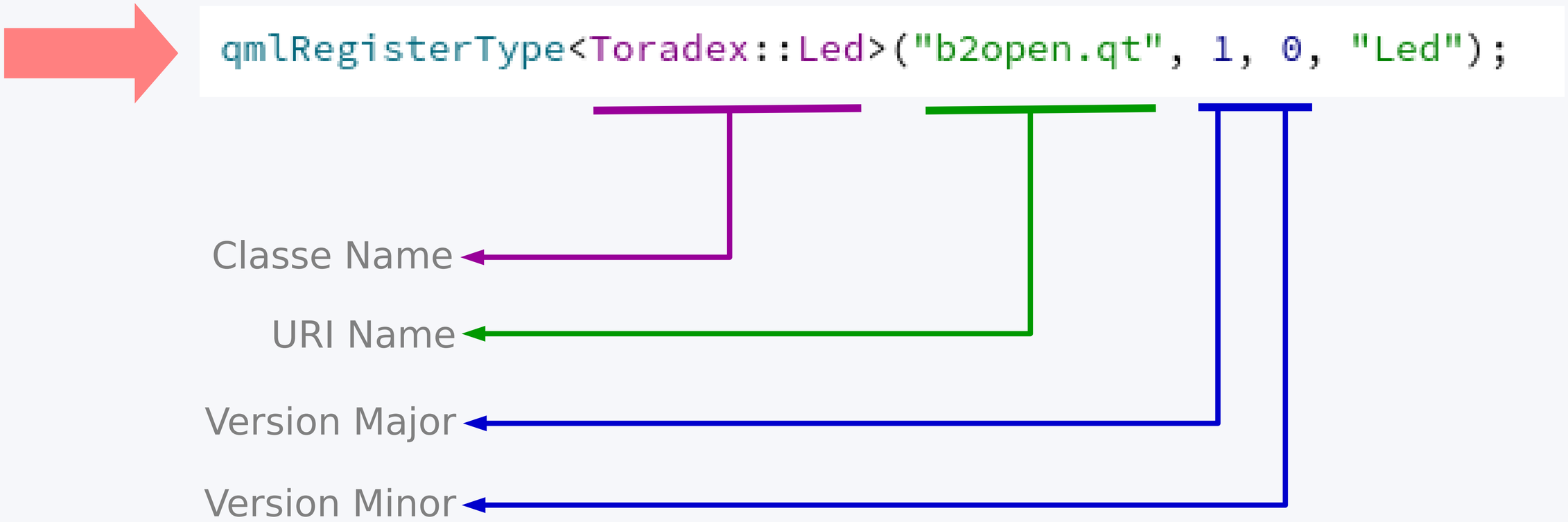
```
qmlRegisterType<Toradex::Led>("b2open.qt", 1, 0, "Led");
```

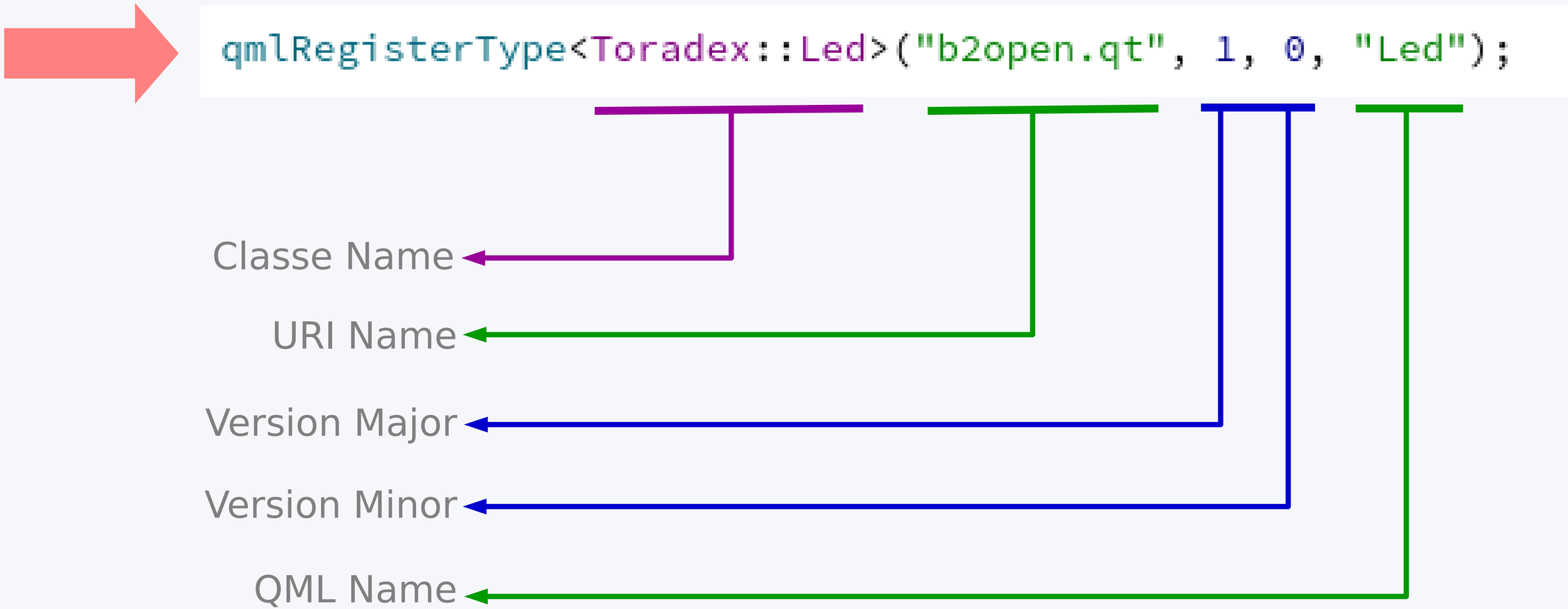


```
qmlRegisterType<Toradex::Led>("b2open.qt", 1, 0, "Led");
```

Classe Name


qmlRegisterType<Toradex::Led>("b2open.qt", 1, 0, "Led");
Classe Name
URI Name


qmlRegisterType<Toradex::Led>("b2open.qt", 1, 0, "Led");
Classe Name
URI Name
Version Major
Version Minor


qmlRegisterType<Toradex::Led>("b2open.qt", 1, 0, "Led");
Classe Name
URI Name
Version Major
Version Minor
QML Name

QML

main.qml

```
import QtQuick 2.9
import QtQuick.Controls 2.3
import b2open.qt 1.0


ApplicationWindow {
    id: root
    visible: true
    width: 640
    height: 480

    Led { id: led101; led: Led.Led101; modo: Led.HeartBeat; }
    Led { id: led103; led: Led.Led103; modo: Led.Timer;     }
    Led { id: led133; led: Led.Led133;
```

C++

main.cpp

```
#include <QDebug>
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "led.h"

int main(int argc, char *argv[])
{
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QGuiApplication app(argc, argv);

    qmlRegisterType<Toradex::Led>("b2open.qt", 1, 0, "Led");

    QQmlApplicationEngine engine;
    engine.load(QUrl(QStringLiteral("qrc:/m    n.qml")));
    if (engine.rootObjects().isEmpty())
        return -1;

    return app.exec();
}
```

led.h

```
#include <QObject>

namespace Toradex {

class Led : public QObject
{
    Q_OBJECT
    Q_PROPERTY(int led                WRITE setLed   NOTIFY changedLed)
    Q_PROPERTY(int modo               WRITE setModo  NOTIFY changedModo)
    Q_PROPERTY(bool ligar  READ ligar WRITE setLigar NOTIFY changedLigar)

public:
    explicit Led(QObject *parent = nullptr);
    ~Led();

    void setLed(const int led);
    void setModo(const int modo);
    void setLigar(bool flag);
    bool ligar() const;
```

QtCon BR

QML

main.qml

```
import QtQuick 2.9
import QtQuick.Controls 2.3
import b2open.qt 1.0


ApplicationWindow {
        root
    ible: true
    th: 640
   ght: 480

    Led { id: led101; led: Led.Led101; modo: Led.HeartBeat; }
    Led { id: led103; led: Led.Led103; modo: Led.Timer;     }
    Led { id: led133; led: Led.Led133;
```

C++

main.cpp

```
#include <QDebug>
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "led.h"

int main(int argc, char *argv[])
{
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QGuiApplication app(argc, argv);

    qmlRegisterType<Toradex::Led>("b2open.qt", 1, 0, "Led");

    QQmlApplicationEngine engine;
    engine.load(QUrl(QStringLiteral("qrc:/m    n.qml")));
    if (engine.rootObjects().isEmpty())
        return -1;

    return app.exec();
}
```

led.h

```
#include <QObject>

namespace Toradex {

class Led : public QObject
{
    Q_OBJECT
    Q_PROPERTY(int led                WRITE setLed   NOTIFY changedLed)
    Q_PROPERTY(int modo               WRITE setModo  NOTIFY changedModo)
    Q_PROPERTY(bool ligar  READ ligar WRITE setLigar NOTIFY changedLigar)

public:
    explicit Led(QObject *parent = nullptr);
    ~Led();

    void setLed(const int led);
    void setModo(const int modo);
    void setLigar(bool flag);
    bool ligar() const;
```

QtCon BR

QML

main.qml

```
import QtQuick 2.9
import QtQuick.Controls 2.3
import b2open.qt 1.0


ApplicationWindow {
        root
    ible: true
    th: 640
    ght: 480

    Led { id: led101; led: Led.Led101; modo: Led.HeartBeat; }
    Led { id: led103; led: Led.Led103; modo: Led.Timer;     }
    Led { id: led133; led: Led.Led133;
```

C++

main.cpp

```
#include <QDebug>
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "led.h"

int main(int argc, char *argv[])
{
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QGuiApplication app(argc, argv);

    qmlRegisterType<Toradex::Led>("b2open.qt", 1, 0, "Led");

    QQmlApplicationEngine engine;
    engine.load(QUrl(QStringLiteral("qrc:/m    n.qml")));
    if (engine.rootObjects().isEmpty())
        return -1;

    return app.exec();
}
```

led.h

```
#include <QObject>

namespace Toradex {

class Led : public Q    ject
{
    Q_OBJECT
    Q_PROPERTY(int led                          WRITE setLed   NOTIFY changedLed)
    Q_PROPERTY(int modo                         WRITE setModo  NOTIFY changedModo)
    Q_PROPERTY(bool ligar  READ ligar WRITE setLigar NOTIFY changedLigar)

public:
    explicit Led(QObject *parent = nullptr);
    ~Led();

    void setLed(const int led);
    void setModo(const int modo);
    void setLigar(bool flag);
    bool ligar() const;
```

QtCon BR

QML

main.qml

```
import QtQuick 2.9
import QtQuick.Controls 2.3
import b2open.qt 1.0


ApplicationWindow {
        root
    ible: true
    th: 640
    ght: 480

    Led { id: led101; led: Led.Led101; modo: Led.HeartBeat; }
    Led { id: led103; led: Led.Led103; modo: Led.Timer;     }
    Led { id: led133; led: Led.Led133;
```

C++

main.cpp

```
#include <QDebug>
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "led.h"

int main(int argc, char *argv[])
{
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QGuiApplication app(argc, argv);

    qmlRegisterType<Toradex::Led>("b2open.qt", 1, 0, "Led");

    QQmlApplicationEngine engine;
    engine.load(QUrl(QStringLiteral("qrc:/m    n.qml")));
    if (engine.rootObjects().isEmpty())
        return -1;

    return app.exec();
}
```

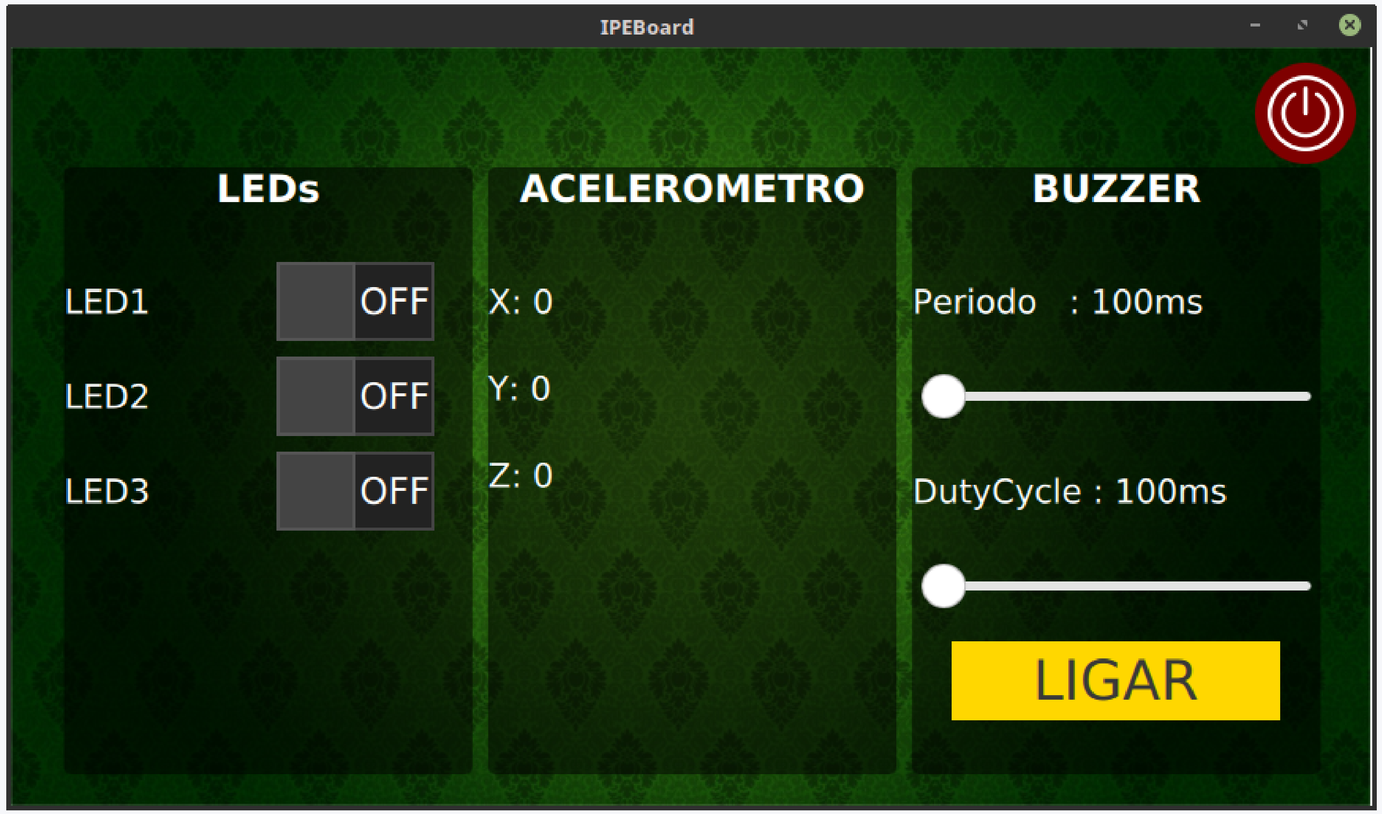
led.h

```
#include <QObject>

namespace Toradex {

class Led : public Q    ject
{
    Q_OBJECT
    Q_PROPERTY(int led                          WRITE setLed   NOTIFY changedLed)
    Q_PROPERTY(int modo                         WRITE setModo  NOTIFY changedModo)
    Q_PROPERTY(bool ligar  READ ligar WRITE setLigar NOTIFY changedLigar)

public:
    explicit Led(QObject *parent = nullptr);
    ~Led();

    void setLed(const int led);
    void setModo(const int modo);
    void setLigar(bool flag);
    bool ligar() const;
```

QtCon BR


Laboratório


B2Weather - B2Open Systems and QtCon2018
29.3°
22.1°
São Carlos
16:29:53


IPEBoard
LEDs
LED1
OFF
LED2
OFF
LED3
OFF
ACELEROMETRO
X: 0
Y: 0
Z: 0
BUZZER
Periodo : 100ms
DutyCycle : 100ms
LIGAR

QtCon BR

- Problemas/Soluções, macetes e dicas valiosas com Qt5
- Parse Command Line
- Tratar Sinais Linux
- Adicionando Fontes ao projeto
- Depurar problemas por debaixo do Qt5

Problema v-sync

No mundo dos gamers o famoso “tearing” o rasgo de tela

#5

Problema v-sync

Exportar no terminal executando a aplicação a variável ambiente **QT_QPA_EGLFS_FORCEVSYNC**.

export QT_QPA_EGLFS_FORCEVSYNC=1

#5

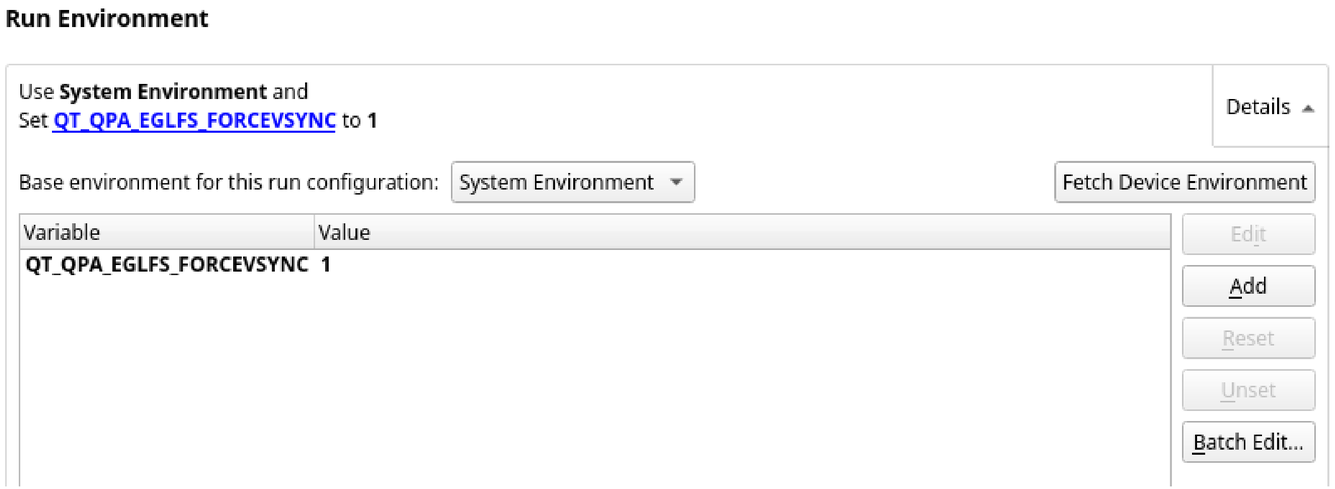
Exportar variável ambiente via QtCreator

Executando a aplicação demo B2Weather do QtCreator(Host) no Target(Kit Toradex).

#6

Exportar variável ambiente via QtCreator

IDE Qt Creator > Projects(Ctrl+5) > Selecionar Kit(ToradexQtCon) > Run Environment



#6

Informações extras sobre Scene Graph

- Qual render loop esta sendo utilizado?
- Informações v-sync
- Fabricante da GPU
- Versão do Driver GPU
- Extensões OpenGL

#7

Informações extras sobre Scene Graph

Exportar no terminal executando a aplicação a variável ambiente **QSG_INFO**.

```
# export QSG_INFO=1
```

#7

Ignorar qDebug()

Muitos desenvolvedores mantêm diversos **qDebug**() em diversas classes durante o desenvolvimento da aplicação, em dado momento não são removidos ou acabam poluindo o stdout durante uma depuração

#8

Ignorar qDebug()

Definir durante a compilação a flag **QT_NO_DEBUG_OUTPUT** no CMake com “add_compile_definitions(**QT_NO_DEBUG_OUTPUT**)” ou adicionando **DEFINE += QT_NO_DEBUG_OUTPUT** no .pro para o QMake processar.

#8

QtCon BR

Obrigado!

perguntas?

Cleiton Bueno
cleiton.bueno@b2open.com