Get Started with Qt for MCUs 1.0

Qt

All in One - Framework

Powerful & Modern Development Framework

Code Once, Deploy Everywhere

Cross-Platform

Integrated Development Tools

Productive development environment

Qt Creator IDE
Design Tools

Target All Your End Users with One Technology

Embedded:

- Embedded Linux, Windows Embedded
- RTOS: QNX, VxWorks, INTEGRITY

Desktop:

- Windows, Linux, macOS
- Enterprise UNIX

Mobile:

- Android, iOS, Universal Windows Platform

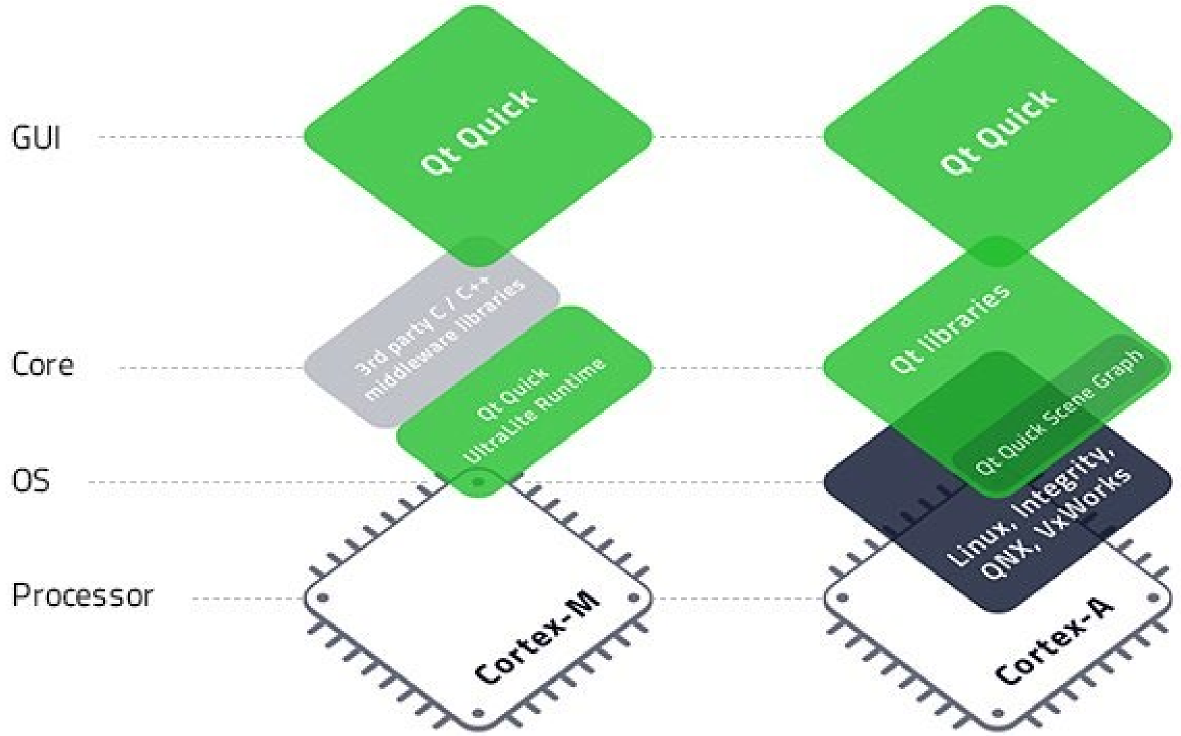
Web:

- WebAssembly

Qt
My Software Service

Qt for MCUs

Ultimate Performance. Tiny Footprint.


GUI
Core
OS
Processor
Qt Quick
3rd party C / C++ middleware libraries
Qt Quick UltraLite Runtime
Cortex-M
Qt Quick
Qt libraries
Qt Quick Scene Graph
Linux, Integrity, QNX, VxWorks
Cortex-A

Qt for MCUs 1.0

Quick Ultralite Graphics runtime

Qt Quick Controls

- Subset of Qt Quick Controls 2.0
- Styling and theming

Platform Adaptation

Bare Metal
- NXP
- STM32
- Renesas

Dev. Tools

- Qt Creator Integration
- Localization (Linguist)

Documentation

Examples with Source Code

Get Started with Qt for MCUs 1.0

Platforms included in the evaluation package:

- **NXP i.MX RT1050 EVK (Bare Metal)**
- **STM32F769i (Bare Metal)**

Software requirements:

- Microsoft Windows
- Qt for MCUs Evaluation Package
- Qt Creator 4.11 or higher
- Qt 5.14
- CMake 3.13 or higher
- Python 2.7 32-bit
- Arm GCC version 8-2019-q3-update or later

NXP:

- Segger J-Link Software Pack
- J-Link OpenSDA RT1050 Firmware

STM32

- STM32CubeProgrammer
- STM32 ST-LINK Utility

Installation

- Qt: https://account.qt.io/downloads
- CMake: https://cmake.org/download/
- Python 2.7 32-bit: https://www.python.org/downloads/release/python-2716/
- Arm GCC: https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads
- J-Link Software Pack: https://www.segger.com/downloads/jlink/JLink_Windows.exe
- J-Link OpenSDA Firmware: https://www.segger.com/downloads/jlink/OpenSDA_MIMXRT1050-EVK-Hyperflash
- STM32CubeProgrammer: https://www.st.com/en/development-tools/stm32cubeprog.html
- STM32 ST-LINK Utility: https://www.st.com/en/development-tools/stsw-link004.html

- Must add in PATH:
  - `<Qt_5.14_install_location>/bin` (for translation tools)

Qt Creator Configuration

- Enable Bare Metal and MCU plugins in Help -> About Plugins (requires a restart)
- Go to Tools -> Options -> Devices
  - Select the ‘MCU’ tab
  - Select the needed board in the ‘Target’ dropdown list
  - Configure the paths of all necessary packages
  - Click ‘Apply’ to generate the Kit
  - (Repeat for each board you want to use)

Qt Creator Configuration (debugging - NXP)

- Add GDB Server Provider
  - Tools -> Options -> Devices -> Bare Metal --> Add -> Default
  - Enter “j-link gdb" as name
  - Set port to 9876
  - Add to init commands:
    ```
    mon reset
    mon halt
    load
    mon reset
    mon halt
    eval "monitor reg pc %#x", &Reset_Handler
    mon go
    ```
  - Add 'mon reset' to reset commands
    ```
    mon reset
    mon halt
    eval "monitor reg pc %#x", &Reset_Handler
    mon go
    ```
- Create Device
  - Tools -> Options -> Devices -> Add…
  - Select ‘Bare Metal Device’
  - Give a name and select the GDB server provider created in the previous step
- Clone the i.MX RT1050 kit created in slide 8 (And rename it to indicate that it is used for debugging)
  - Change the device type to Bare Metal Device
  - Select the device created in the previous step

Qt Creator Configuration (debugging – STM32)

- Add GDB Server Provider
  - Tools -> Options -> Devices -> Bare Metal --> Add -> Default
  - Enter “j-link gdb" as name
  - Set port to 9876
  - Add to init commands:

    ```
    load
    ```
  - Add 'mon reset' to reset commands

    ```
    mon reset
    ```
- Create Device
  - Tools -> Options -> Devices -> Add…
  - Select ‘Bare Metal Device’
  - Give a name and select the GDB server provider created in the previous step
- Clone the STM32F769i kit created in slide 8 (And rename it to indicate that it is used for debugging)
  - Change the device type to Bare Metal Device
  - Select the device created in the previous step

Create Project

- New Project -> Application-> Mcu Support Application
  - <u>Make sure to not have spaces in the project path or flashing won’t work</u>
  - Select the Kit corresponding to your board
- Open CMakeLists.txt
  - Add ‘C’ to the languages argument in the line, as such:
    `project(<myproject> VERSION 0.0.1 LANGUAGES C CXX)`

Flash and Run on Board

- Click *Run* (green play button) in Qt Creator

Or

- NXP: Using Segger J-Flash Lite utility
  - Select MIMXRT1052DVL6B in device list then ‘OK’
  - In ‘Data File’, select the .hex file from the project’s build directory
  - Click “Program Device”
- STM32: Using ST-LINK Utility (or CubeProgrammer)
  - External Loader -> Add External Loader
  - Select 'MX25L512G_STM32F769I-DISCO' and validate
  - Target -> Connect
  - Target -> Program & Verify
  - Open built .hex file and click ‘Start’

Develop - Basics

- Root Rectangle
  - Set color to "#f0f3f4"
  - Setting the size is optional, the UI will fill the screen by default
- Text
  - Change `color` property to “`black`”
  - Set `font.pixelSize: 24`

Develop – Fonts

To use custom fonts:

- Copy font files to a 'fonts' directory inside the project
- In CMakeLists.txt, add:

```
set(QUL_FONTS_DIR "${CMAKE_CURRENT_SOURCE_DIR}/fonts")
set(QUL_DEFAULT_FONT_FAMILY "<name_of_your_default_font_family>")
```

- If multiple fonts are used, configure which font to use in each Text item with:

```
font.family: "<name_of_your_default_font_family>")
```

Develop – Layout and Images

- Layout: Add `Column` and set `spacing` property to `16`
  - Set `anchors.centerIn: parent`
- In Text: replace `anchors.centerIn` with `anchors.horizontalCenter: parent.horizontalCenter`
- Image
  - Add and image file to the project directory (it can be in a subdirectory)
  - Add `Image` in QML file
    - Set `source` property to the relative path of the image e.g. “`images/logo.png`”
    - Set `anchors.horizontalCenter: parent.horizontalCenter`

Develop - Controls

- Add `Qul::QuickUltraliteControlsStyleDefault` in `target_link_libraries` in CMakeLists.txt
- If you want to customize the controls' look & feel, read the '*Qt Quick Ultralite Controls styling*' documentation page
- Add QtQuick.Controls import
- Add `Switch` in qml file
  - Set `anchors.horizontalCenter: parent.horizontalCenter`
  - `id: switchButton`
- In the `Image`
  - `opacity: switchButton.checked ? 1 : 0`

Develop - Animations

- Add to Image: *`Behavior on opacity { NumberAnimation { duration: 600; easing.type: Easing.OutCubic } }`*
- Wrap Image inside an Item
  - Set `id: logo` inside Image
  - In Item:
    - `width: logo.width`
    - `height: logo.height`
    - `anchors.horizontalCenter: parent.horizontalCenter`
  - Remove anchors from Image
- Add to Image
  - *`y: switchButton.checked ? 0 : 50`*
  - Duplicate behavior line, but apply to ‘y’

Develop – Business Logic in C++

- Business logic and HW access is implemented in C++
- Add your .h and .cpp to CMakeLists.txt with `target_sources(<project_name> PRIVATE …)`
- Create a QML wrapper for your C++ APIs:
  - File -> New -> C++ -> C++ Header File
  - Add to *target_ sources() in* CMakeLists.txt

Develop – Business Logic in C++

- Add `#include <qul/singleton.h>` or `<qul/qtobject.h>`
- Add
  ```
  struct YourWrapper : public Qul::Singleton<YourWrapper>
  or
  struct YourWrapper: public Qul::Items:QtObject
  ```
- Add your properties, functions and signals (see documentation)
- In CMakeLists.txt
  - `qul_target_generate_interfaces(<projectname> your_wrapper.h)`
  - `target_include_directories(<projectname> PUBLIC ${CMAKE_CURRENT_SOURCE_DIR})`
  - `target_include_directories(<projectname> PUBLIC ${CMAKE_CURRENT_BINARY_DIR})`
- You can instantiate YourWrapper in QML if using QtObject or directly use YourWrapper.someFunction() if using Singleton.

Develop – Translations

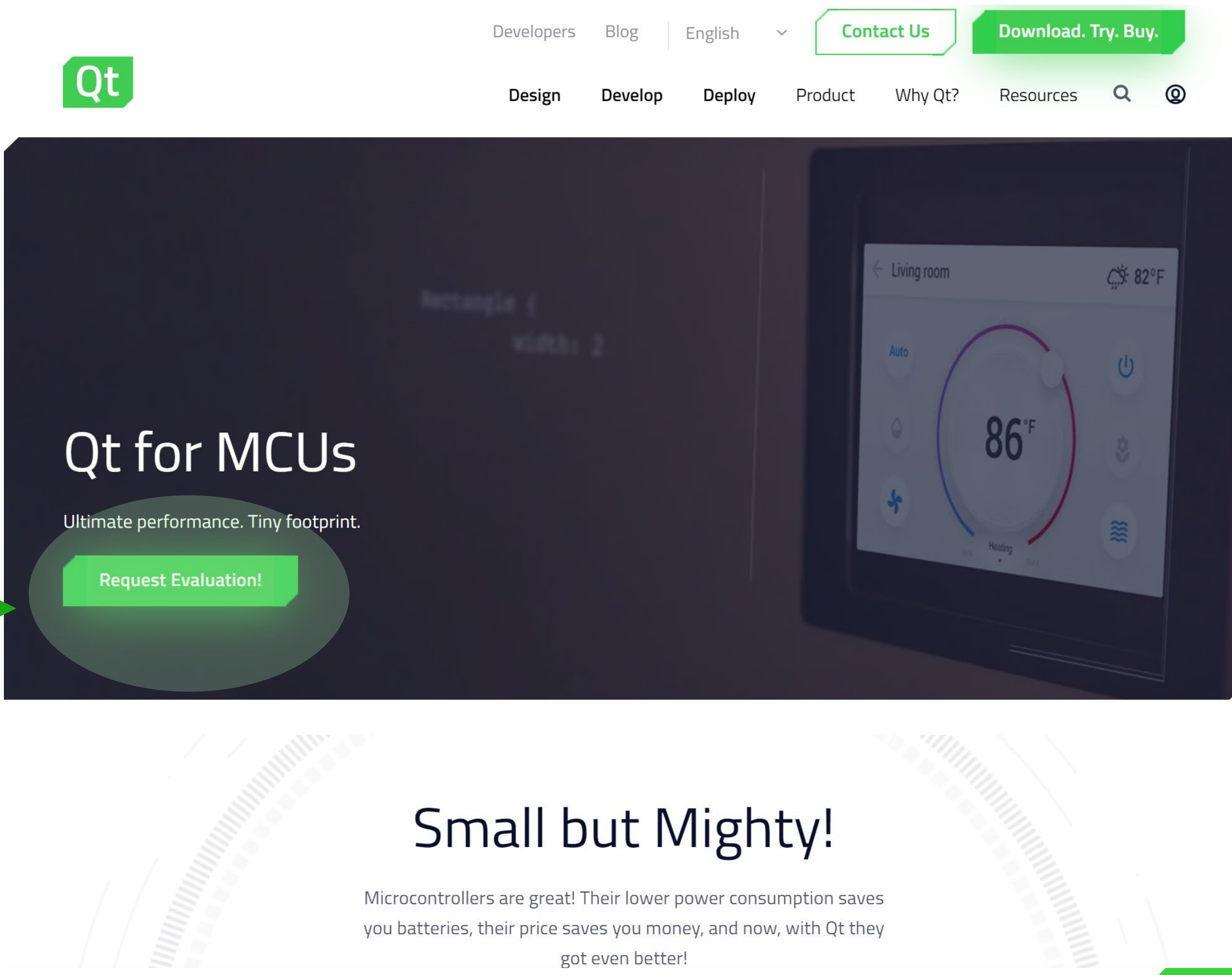
- Wrap all your translatable strings with `qsTr()` in QML files
- In CMakeLists.txt
  - `qul_target_embed_translations(<project_name> <project>.<language_code>.ts)`
    example file name: myproject.en_US.ts
  - Add as many filename.ts as languages you need to support
- Generate .ts file
  - Projects -> Add build configuration (release)
  - Rename to "Update Translations"
  - Click on 'Details' in 'Build Steps'
  - Select the 'update_translations' target
  - Build to generate the .ts files
  - The "Update Translations" target needs to be re-run every time you add or modify any occurrence of `qsTr()`
- Open .ts files with Linguist to translate the strings
- The active runtime language can be changed with:
  `Qul.uiLanguage = "<language_code>" // or "source" to use the language used in the source code`

Debug

- Start GDB Server
  - Open Command Prompt
  - NXP:
    - "C:\Program Files (x86)\SEGGER\JLink\JLinkGDBServer.exe" -device MCIMXRT1052 -if SWD -scriptfile <Qt_for_MCUs_install_dir>\CMake\evkbimxrt1050\evkbimxrt1050_sdram_init.jlinkscript -port 61234
  - STM32
    - cd C:\ST\STM32CubeIDE_1.0.2\STM32CubeIDE\plugins\com.st.stm32cube.ide.mcu.externaltools.stlink-gdb-server.win32_1.0.0.201904160814\tools\bin
    - .\ST-LINK_gdbserver.exe -l 31 -p 61234 -r 15 –d -e -cp "C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin" -el "C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\ExternalLoader\MX25L512G_STM32F769I-DISCO.stldr"
- Use GDB CLI
  - Open Command Prompt
    - cd <arm_gcc_install_dir>\bin
    - arm-none-eabi-gdb.exe "<your_project_build_directory>\Debug\<your_project>.elf"
    - target extended-remote localhost:61234
- Or use the Bare Metal plugin in Qt Creator
  - Before debugging, select the kit that uses the Bare Metal Device (configured in slide 9-10)
  - Start debugging

Thank You!

Give it a try!
https://www.qt.io/qt-for-mcus



Qt for MCUs

Ultimate performance. Tiny footprint.

Small but Mighty!

Microcontrollers are great! Their lower power consumption saves you batteries, their price saves you money, and now, with Qt they got even better!