ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

How to build and use an SDK for QT

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

A quality version of this page, approved on *4 December 2020*, was based off this revision.

The OpenSTLinux distribution enables the generation of example images based on the QT framework (st-example-image-qt and, for ecosystem release v2.1.0 , st-example-image-qtwayland) for demonstration purposes, not for developing products.

This article provides information on how to create the SDK for building QT applications based on the above example images.

Contents

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

1 Prerequisites

Be able to rebuild an OpenSTLinux image

2 OpenSTLinux QT images and SDK

OpenSTLinux distribution provides two examples of images based on the QT framework:

- st-example-image-qtwayland (requires 'openstlinux-weston' distro and ecosystem release v2.1.0): thanks to weston window management, QT applications are displayed in independent windows and can coexist on the same screen with other non-QT applications;
- st-example-image-qt (requires 'openstlinux-eglfs' distro): only one QT application can access the screen. It is displayed in full-screen mode.

To build the images and the associated SDK, first install the OpenSTLinux distribution by following the procedure described in chapter Installing the OpenSTLinux distribution.

2.1 QT image and SDK with weston/wayland

The QT image and SDK use QTwayland back-end to run QT applications in independent windows managed by weston. This requires ecosystem release v2.1.0 .

Initialize the OpenEmbedded build environment for the openstlinux-weston distro:

```
PC $> DISTRO=openstlinux-weston MACHINE=stm32mp1 source layers/meta-st/scripts/envsetup.sh
```

Read and accept EULA. Refer to Initializing the OpenEmbedded build environment for further details.

Then build the image and the SDK:

```
PC $> bitbake st-example-image-qtwayland
PC $> bitbake st-example-image-qtwayland -c populate_sdk
```

Refer to Generating your own Starter and Developer Packages and How to create an SDK for OpenSTLinux distribution for further details.

Note: the legacy command **bitbake meta-toolchain-qt5** is not recommended because the resulting SDK would miss some target packages.

The image can be flashed on the target board as in Flashing the built image.

The generated SDK is available in the **tmp-glibc/deploy/sdk/** folder. It can be installed as described in Run the SDK installation script, using the following command:

```
PC $> ./tmp-glibc/deploy/sdk/st-example-image-qtwayland-openstlinux-weston-stm32mp1-
x86_64-toolchain-3.1-snapshot.sh -y -d <working directory absolute path>/Developer-
Package/SDK
```

2.2 QT image and SDK with EGLFS

The QT image and SDK use QT back-end to run single QT applications in full-screen mode.

Initialize the OpenEmbedded build environment for the openstlinux-eglfs distro:

```
PC $> DISTRO=openstlinux-eglfs MACHINE=stm32mp1 source layers/meta-st/scripts/envsetup.sh
```

Read and accept EULA. Refer to Initializing the OpenEmbedded build environment for further details.

Then build the image and the SDK:

```
PC $> bitbake st-example-image-qt
PC $> bitbake st-example-image-qt -c populate_sdk
```

Refer to Generating your own Starter and Developer Packages and How to create an SDK for OpenSTLinux distribution for further details.

Note: the legacy command **bitbake meta-toolchain-qt5** is not recommended because the resulting SDK would miss some target packages.

The image can be flashed on the target board as in Flashing the built image.

The generated SDK is available in the **tmp-glibc/deploy/sdk/** folder. It can be installed as described in Run the SDK installation script, using the following command:

```
PC $> ./tmp-glibc/deploy/sdk/st-example-image-qt-openstlinux-eglfs-stm32mp1-x86_64-
toolchain-3.1-snapshot.sh -y -d <working directory absolute path>/Developer-Package/SDK
```

2.2.1 Selecting the display resolution and size in EGLFS

When using a display that accepts multiple resolutions, the preferred resolution can be specified by editing the board file **/usr/share/qt5/cursor.json**. For example, for an HDMI display, change the line

```
{ "name": "HDMI1", "mode": "1280x720" },
```

and enter one of the valid resolutions reported by the command

```
Board $> modetest
```

If the current display resolution is higher than the resolution requested by the QT application, QT expands the application to run it in full screen. This can produce blurred images on the display and can impact the system performance.

The QT framework can be configured to use only one part of the overall display. For example, to use only an area of 400x300 pixels, add:

```
{ "name": "HDMI1", "mode": "1280x720", "size": "400x300" },
```

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

3 Configuring the QT framework

By default, QT internal data use 64 bits per pixel (16 bits for each R, G, B and A component). The performance can be slightly improved by forcing the QT framework to use 32 bits per pixel (8 bits for each component). This can be achieved by editing the file **layers/meta-st/meta-st-openstlinux/recipes-qt/qt5/qtbase_git.bbappend** before the build and replacing the line

```
QT_CONFIG_FLAGS += " -no-sse2 -no-opengles3"
```

with

```
QT_CONFIG_FLAGS += " -no-sse2 -no-opengles3 -no-feature-raster-64bit"
```

Note: the above setup can produce visible artefacts and has to be evaluated on a case by case basis.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

4 Building a QT application

Open the folder that contains the application, enable the QT SDK and compile the application:

```
PC $> cd <path_of_app>
PC $> . <install_path_sdk>/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
PC $> qmake && make
```

Software development kit (A programming package that enables a programmer to develop applications for a specific platform.)

High-Definition Multimedia Interface (HDMI standard)