Overview

On this website you can find the official public documentation for the Linux-based software BSPs and the demo hardware from **Kontron Electronics GmbH** (formerly exceet electronics GmbH).

In the menu on the left side you can find some general information about the Kontron BSPs and boards.

For more specific information about the platform of your choice, please use the **dropdown menu on the top of the page** to switch to the documentation of a **platform/BSP-version**.

Prerequisites

In most cases, the documentation covers only the **very basic aspects** of how to **use, build, configure and setup** the hardware and software for general purposes. We assume, that the reader has some **background knowledge** in the area of **embedded Linux** and it might be needed to consult other ressources.

Company Name

This documentation uses the acronym **"KED"** (which means **"Kontron Electronics Deutschland"**) to refer to **"Kontron Electronics GmbH"**, which is part of "Kontron S&T AG".

Coustomized Hard- and Software

Coustomized hard- and software is not part of this documentation. Please consult Kontron Electronics for information about your customer-specific project.

Open-Source

Most parts of our software and documentation are **open-source**. We support the idea behind FOSS and actively contribute to projects such as **Linux, U-Boot and Yocto/OE**.

Contribute

If you work with our hard- and software or with the documentation, we would like to encourage you to **contribute** by **reporting issues** or sending **merge requests**. Please see Contribute for further information on how to do so.

Software Licensing

Company Name

As for the other parts of the documentation, this page uses the acronym **"KED"** (which means **"Kontron Electronics Deutschland"**) to refer to **"Kontron Electronics GmbH"**, which is part of "Kontron S&T AG".

Completeness and Correctness

The information on this page does not make any claim to be complete or to be legally correct. It is intended to give a basic idea of licensing how we at KED understand it and can serve as a starting point for your own considerations.

Licenses of Software Packages

The software delivered with boards and modules (Board Support Package, BSP) by Kontron Electronics GmbH contains open-source software with license agreements that, among other things, restrict linking against closed-source applications (e.g. GPL, LGPL). Before using any of the libraries or applications, it is therefore necessary to check the license agreements of the used source code. The licenses are contained within the source code of the software packages. Furthermore it is necessary to check any valid patents and license conditions of the used software, especially for multimedia formats (e.g. mp3 format). Kontron Electronics GmbH does not assume any liability for infringements of patents or license agreements of parts of the provided BSP. Inside the Yocto build system you find a directory in `<builddir>/tmp/deploy/licenses`, that holds copies of all the licenses of the packages, that were built. To get a list of all the packages and their licenses included in your image, you can look at the file `license.manifest` in `<builddir>/tmp/deploy/licenses/<full-image-name>`.

If you have no access to those files, feel free to ask KED to provide them for you.

Typical open-source licenses

Here are some notes and further information on different licenses.

GPLv2 used by the Linux kernel and many other packages

If you release or redistribute a product, that includes software under GPLv2, you are bound to provide the source code of those parts with your product (copyleft). You can either include the source code with your product and deliver it together, or you can include a written offer to provide the source code when requested.

- Attention: GPLv2 does not allow you to provide the source code via a network service (e.g as download). You must deliver it on physical media. Only GPLv3 allows delivery via downloads.
- Many GPLv2 licensed packages include the possibility to license it under a later version of the same license (e.g. GPLv3), but the Linux kernel for example is GPLv2 only.
  If you use code inside your application that is licensed under the GPL you must not keep your application code closed, you are obligated to use the GPL and have to provide the programs source code to your customers.
- Frequently Asked Questions (FAQ)
- A Practical Guide to GPL Compliance

GPLv3

The GPLv3 is the modernized and updated version of the GPLv2.
See http://www.gnu.org/licenses/quick-guide-gplv3.html.

LGPLv2.1 used by many libraries

The LGPL is similar to the GPL, but allows that your own proprietary applications link against libraries that are licensed under LGPL, without the need to make your application code public. If you make changes or additions to the original software, you have to provide these changes to your customer.

LGPLv3

The LGPLv3 is the modernized and updated version of the LGPLv2.

MIT

The MIT license even allows you to modify and distribute software packages, without the need to publish the source code (no copyleft). It is still necessary to include the license notice in your product.

BSD

The BSD license is similar to the MIT license and has no copyleft.

Proprietary licenses (e.g Freescale/NXP or other HW manufacturers)

The BSP might also contain packages, firmware or drivers, that are licensed under proprietary licenses by the manufacturer or other third parties. Depending on your product, those agreements need to be checked for compliance.

License compliance

It is essential to make sure, that your final product complies with all the licenses of the included software packages. As mentioned in the last paragraph it depends on the used licenses what you have to do for a full license compliance.

Notification in the manual

Almost all licenses require that you inform your customers in your manual that you use open-source software. You have to mention that parts of your software are open-source software and deliver a list of the components you are using and their particular licenses.

Source code delivery

If you are using software licensed as GPL or LGPL you have to deliver the source code and license texts of this software to your customers or at least, make it possible that your customer can get it. There a several options to achieve this.

- Direct delivery with your product: You can accompany your product with a volume containing all used source code under the mentioned licenses.
- Written offer: You can state in your manual that every customer of your product can get a copy of the source code as long as the delivery of your product is no longer ago than three years or as long as you deliver spare parts. It is allowed to ask for a small fee to cover your expenses.
- Download (For (L)GPL V3 code only): You can send your customers a link where they can download the source code. You have to guarantee that the link will be accessible for the same time as it would be for the written offer.

  KED will automatically generate an archive with the used source code files of your product. Due to the size of this file of a few GB we will not send it to you every time it will be generated.
  Please feel free to contact KED whenever you need this archive. Please be aware that we only can integrate the source code of the programs we have access to. If you add further open-source software, you have to append these sources to our archive.

Adding own software to your product

If you are using (L)GPL in version 3 licensed code you have, for version 2 it is recommended, to give your customer the opportunity to install his own programs on your product. For code under (L)GPL version 3 you are obligated to grant your customer the right to install his own programs on your product. Version 2 of the license only recommends, but does not enforce this. To ensure the security of your device, this opportunity does not have to be included in your product from the beginning, it is appropriate to require your customer to send the product back to you and you will disable the necessary security features to allow the installation of custom programs. You can demand that all

given warranties of your product will expire at the time of applying custom software.

Exemplary text for your manual:

English:

```
This product contains software components which are licensed as
free
respectively open-source software under the GNU General Public
License, versions
2 or 3, or the GNU Lesser General Public License, versions 2.1 or
3.
Everyone can get the source code of this software components from
us on a data
storage medium (CD-ROM, DVD, USB drive) if requested at our
customer support at
the following address within three years after the delivery of the
product or as
long as we offer spare parts or support for the product.

[Name of the company]
[Contact person]
[Address]

Including the statement of the following product data:

[Product name]
[Serial number]
[Date of delivery]

We also require a fee of EUR 10,- for the costs of preparation of
the medium and
shipping to be transferred to the following bank account [Bank
account]
Preventive it should be mentioned here that using the right of
installing own
versions of the open-source software components, which is
guaranteed in the
license contract, will expire all certifications and warranties of
the product.
The operation of the manipulated product will happen on one's own
authority.
```

German:

```
Dieses Produkt enthält Softwarebestandteile, die von den
Rechteinhabern als
Freie Software bzw. Open Source Software unter der GNU General
Public License,
Versionen 2 bzw. 3, bzw. der GNU Lesser General Public License,
Versionen 2.1
bzw. 3.0, lizenziert werden. Jedermann kann den Quellcode dieser
Softwarebestandteile von uns auf einem Datenträger (CD-ROM, DVD
oder USB-Stick)
erhalten, wenn innerhalb von drei Jahren nach der Auslieferung des
Produkts an
den Kunden oder solange, wie wir Ersatzteile oder Support für das
Produkt
anbieten, eine Anfrage an unsere Kundenbetreuung an folgende
Adresse

[Name der Firma]
[Ansprechpartner]
[Adresse]

mit Angabe folgender Produktdaten

[Name]
[Seriennummer]
[Auslieferungsdatum]

gestellt wird und EUR 10,- für die Kosten zur Erstellung des
Datenträgers und
dessen Versendung vorab auf folgendes Konto [Kontoverbindung]
überwiesen werden.
Vorsorglich wird darauf hingewiesen, dass die Nutzung des im
Lizenzvertrag
zugesicherten Rechts, die Open Source Komponenten gegen eigene
Versionen
auszuwechseln, zum Erlöschen der Zertifizierung bzw. Garantie
führt. Der Betrieb
des entsprechend geänderten Gerätes erfolgt auf eigene
Verantwortung.
```

Building Yocto image from source code

Install Yocto as described here: https://www.yoctoproject.org/docs/current/mega-manual/mega-manual.html Unpack the archive in the yocto main folder. Setup your environment and download the missing code to build the image. Attention: Some of the code could not be downloaded as it has a proprietary license. Remove these components of the image.
If you have questions, please contact support@kontron-electronics.de.

Using Qt in a Product

There are several licensing options for Qt. You should decide on one of them before starting to design your application, as switching from the open-source license to the commercial license is not allowed.

Here are some sources for further information:
Qt Licensing
Qt: Making the right licensing decision

Git Server Overview

The relevant repositories are hosted on a public GitLab server at https://git.kontron-electronics.de. You can find all public repositories **here**.

The software environment consists of several parts described below.

KED Yocto Core Repository

```
Web-URL: https://git.kontron-electronics.de/yocto-ktn/yocto-ktn
Git-URL: https://git.kontron-electronics.de/yocto-ktn/yocto-ktn.git
```

The Yocto core repository bundles some documentation and useful scripts to initialize and manage Yocto-builds for Kontron hardware. Cloned to your local file system, the `yocto-ktn` directory is used as a root directory for the build environment.

Directory Tree (summary)

```
yocto-ktn
│
├── docs          # some generic documentation sources (e.g. for
this page)
├── layers        # empty directory to store meta-layers
├── scripts       # scripts to automate certain tasks
└── init-env      # this is a script to initialize the build
environment
```

KED Yocto Build Repositories

The build repositories contain the configuration for a specific Yocto build. It defines the layers and revisions used. For a list of available build repositories, look for the projects prefixed with `build-` here. The build repositories are cloned to the `yocto-ktn` directory.

After finishing the build, the build directory also holds the `tmp` subdirectory with all the intermediate and final build results and products.

Directory Tree (summary)

```
build-*
|
├── ci                  # scripts and config for the KED CI builds
├── conf                # configuration directory
|   ├── local.conf      # contains local build settings (usually
not tracked by git)
|   ├── bblayers.conf   # defines the directories parsed by bitbake
|   └── repo.conf       # defines the repositories handled by 'init-
env' and
|                       # 'meta-*' scripts
|
└── docs                # build-/platform-specific documentation
sources
```

KED Yocto Meta Layers

The KED Yocto meta layers, together with other layers provided by the SoC vendors and the OE community, contain the actual metadata for everything that is put into the resulting BSP.

In the KED build system all layers are put into the `yocto-ktn/layers` directory.

meta-ktn

This layer contains configuration for the `ktn` distribution, generic recipes for KED applications, modifications and appends.
For all BSP versions based on the `Rocko` -branch and earlier, this layer is the only layer provided by KED and contains only i.MX6-based machines. Later BSP releases use a different scheme (see below).

```
Web-URL: https://git.kontron-electronics.de/yocto-ktn/meta-ktn
Git-URL: https://git.kontron-electronics.de/yocto-ktn/meta-ktn.git
```

Directory Tree (summary)

```
meta-ktn
|
├── classes           # OE Metadata classes (.bbclass)
├── conf              # OE Metadata configuration (layer, distro,
machines)
├── freescale-layer   # OE Metadata depending on the meta-
freescale layer
├── networking-layer  # OE Metadata depending on the meta-
networking layer
├── qt5-layer         # OE Metadata depending on the meta-qt5 layer
├── swupdate          # OE Metadata depending on the meta-swupdate
layer
└── recipes-*         # OE Metadata recipes (.bb, .bbappend)
```

meta-ktn-* (only for BSPs based on "Thud"-branch and later)

From BSP releases based on the "Thud"-branch and later, `meta-ktn` is only used for platform-agnostic meta data and additional layers prefixed with `meta-ktn-` are used for platform-specific support (e.g. machine configurations).

meta-ktn-imx

```
Web-URL: https://git.kontron-electronics.de/yocto-ktn/meta-ktn-imx
Git-URL: https://git.kontron-electronics.de/yocto-ktn/meta-ktn-
imx.git
```

meta-ktn-stm32mp

```
Web-URL: https://git.kontron-electronics.de/stm32mp/meta-ktn-
stm32mp
Git-URL: https://git.kontron-electronics.de/stm32mp/meta-ktn-
stm32mp.git
```

Directory Tree (summary)

```
meta-ktn-*
|
├── conf         # OE Metadata configuration (layer, distro,
```

```
machines)
├── qt5-layer    # OE Metadata depending on the meta-qt5 layer
├── swupdate     # OE Metadata depending on the meta-swupdate layer
├── recipes-*    # OE Metadata recipes (.bb, .bbappend)
└── wic          # configuration for the OE Image Creator (Wic)
```

KED Customer Repositories

The default repositories to build BSPs for demo and evaluation hardware are public and require no authentication. Customer-specific repositories might be private and need an user account on the GitLab server. To gain access to such repositories please request a useraccount on the Kontron Electronics GitLab server. We recommend to use SSH and authentication with keys.

Documentation

The generic public documentation can be found in the `yocto-ktn` repository (see above). Further platform-specific documentation are located in the `docs` directories inside the corresponding build repositories (see above).

All documentation is written in the `Markdown` markup language and is rendered to HTML by `mkdocs` .

Yocto Build System

This chapter explains the steps necessary to setup and use the Yocto-based build system in order to generate BSP images for Kontron hardware. After setting up the system you can start a build and get a bootloader, kernel image and root filesystem to run on your target hardware.

This is a generic guide, that only describes the basic setup. For practical examples and more details for your platform of choice, please see the platform-specific guides:

- KED Yocto Build System for NXP i.MX
- KED Yocto Build System for STM32MP1

Please note that Kontron Electronics might also provide you with a pre-configured virtual machine image. If you use this, the build system is already installed and you can immediately start to build an image for your target device.

Ubuntu 18.04 LTS 64-bit is used as reference OS for the development PC.

Installing Prerequisites

If you start from the beginning, it might be necessary to install some prerequisites on your development PC. Therefore do

```
sudo apt update
```

to update your package index. Afterwards start the package manager *apt* to install the required packages:

```
sudo apt install git-core gcc g++ python gawk chrpath texinfo
libsdl1.2-dev \
                 gdb-multiarch gcc-multilib g++-multilib
```

Please also see the official Yocto docs for additional packages, that might be needed.

GitLab Server and Repositories

For an overall overview of the server and the available reposiories, please consult the "Git Server Overview" page.

Gaining Access to Private Repositories on the KED GitLab Server

> **Skip if you only intend to use public repositories**
>
> All the repositories that are not customer- or project-specific are publicly accessible.
> Therefore to get started with an Eval-Kit or generic Kontron hardware, you can skip this step

Generating a SSH-key on your Machine

First check if you already have an existing SSH-key in `~/.ssh/` (id_rsa and id_rsa.pub). If yes you can use it in the next step. If not use the following commands to generate a key:

```
mkdir ~/.ssh
chmod 700 ~/.ssh
ssh-keygen -t rsa
```

You can add a passphrase for additional security when prompted. For more information on SSH authentication please visit the Ubuntu Help.

Adding the SSH-key to your GitLab Account

In the top right corner click on your profile picture. Click "Settings" and navigate to "SSH Keys" in the left navigation. Copy and paste your key and give it a name (e.g. work-pc). Copy the content of your `ida_rsa.pub` file from the previous step and paste it in the "Key" input field. Click "Add Key".

Repository and Directory Structure

This is how the directory tree with the most important files and directories will look like:

```
yocto-ktn                   # the core repository
|
├── build-foo               # a build repository
|   |
|   ├── conf
|   |   ├── repo.conf       # specifies the revisions of all layers
|   |   ├── local.conf      # specifies local settings for the
build
|   |   └── bblayers.conf   # specifies all layers that will be
parsed by bitbake
|   |
|   └── tmp                 # contains all of the build data
|       ├── deploy
|       |   ├ images        # contains image files and binaries
for the target
|       |   ├ ipk           # contains packages
|       |   ├ licenses      # contains licenses of the packages in
use
|       |   └ sdk           # contains SDK and toolchain binaries
|       |
|       └── work
|           └ ...           # contains all source and build files
for the packages
|
├── layers                  # contains all meta layers with recipes
|   |                       # (each one is a git repository)
|   |
|   ├ poky                  # contains the Yocto/Poky build system
and meta data
|   ├ meta-openembedded     # contains basic meta layers
|   ├ meta-ktn              # contains basic Kontron adaptations
and modifications
|   ├ meta-ktn-xyz          # contains Kontron platform
adaptations and modifications
|   └ ...
|
├── scripts                 # contains scripts to automate certain
tasks
├── downloads               # contains all the files downloaded by
the fetcher
|                           # (shared by all builds)
```

```
├── sstate-cache            # contains the sstate cache (shared by
all builds)
└── init-env                # this is a script to initialize the
build environment
```

Cloning

Cloning the Core repository (yocto-ktn)

To clone the necessary repositories for your build, go to a directory on your system where you want all the data needed (including source files, build, cache, config, etc.) to be saved (usually $HOME). Please note, that - depending on your build - this usually requires a lot of disk space (> 50 GB). If you have to choose between a SSD and a HDD for running the build, use the SSD as this gives you a little extra speed.

```
cd ~
```

When using SSH access, add the Kontron Electronics GitLab server to the list of known SSH hosts on your machine by running:

```
ssh git@git.kontron-electronics.de
```

Clone the main repository (yocto-exceet). Please note, that the subdirectory yocto-exceet is created automatically.

```
git clone https://git.kontron-electronics.de/yocto-ktn/yocto-
ktn.git
```

Cloning Additional Build Repositories

Customer-specific data like kernel configurations, devicetrees for custom boards, custom recipes, etc. is kept in a separate meta-layer within the yocto-ktn system. Customer-specific build configurations are also kept in a separate build directory. The default build configurations for Kontron Electronics Eval-Kits and standard hardware are also kept in repositories like `build-ktn-imx` or `build-stm32mp`.

The most convenient way to initialize a build and clone all necessary repositories is by using the init-env script. Run this script with the desired build configuration (name of the build repo) as argument. See Initializing the build environment.

Initializing the Build Environment

Before being able to build an image, the metadata for all components needs to be fetched. This is usually done through initializing the environment by sourcing the `init-env` script.

By default this script also runs the `meta-update` script (see Updating the repositories).

Metadata and Repository Overview

For an overview of the repositories and a summary of the metadata inside them, please have a look at the "Git Server Overview" page.

Sourcing the init-env script automates the following tasks:

1. Running the `meta-update` script from `yocto-ktn/scripts/`
    a. Updating the core repository (yocto-ktn) to the latest revision
    b. Cloning/Updating the build-repository (only if `-u` option is used)
    c. Parsing the file `conf/repo.conf` in the build directory
    d. Cloning/Updating all meta layers to the revisions from `repo.conf`
2. Running the `oe-init-build-env` script from `layers/openembedded-core`
    a. Initialize the build environment for `Bitbake`
    b. If no `conf/local.conf` file exists in the build-directory, create one from the template
3. Selecting a machine if the `-m` option is used, by setting the environment variable `MACHINE`.

For other options of init-env and meta-update, please run `. init-env -h` or `meta-update -h`.

By sourcing `init-env` you also change to the build directory and therefore you are ready to run the `bitbake` command.

Examples

Here are some examples for intializing different kinds of builds.

```
cd ~/yocto-ktn
```

To Init the 'build-ktn-imx' for the 'kontron-mx6ul' Machine:

```
. init-env -m kontron-mx6ul build-ktn-imx
```

> **Important**
>
> When building for the first time you additionally have to set the option `-u`

To use a custom `build-<customer>`:

```
. init-env build-<customer>
```

To use a custom `build-<customer>` with a specific Yocto BSP branch (only if multiple branches such as thud, warrior, etc. are available):

```
. init-env -r <BSP branch> build-<customer>
```

To initialize the environment and update to the latest revision of the build repository, use the `-u (update)` option:

```
. init-env -u build-<customer>
```

To initialize the environment and skip checkout errors (e.g. when you have local uncommited changes in some layer), use the `-s` option:

```
. init-env -s build-<customer>
```

After initializing, you are ready to build a recipe, a complete image or the sdk for your machine with the Yocto `Bitbake` tool. See Using Bitbake for further information.

Updating the Repositories

As time goes by new versions of the used layers may be available. Updating the repositories is conveniently done by running the `meta-update` script in `yocto-exceet/scripts` . However this is often not necessary, because it is automatically run while initializing the build environment.

The `meta-update` script tries to fetch the most recent versions of the core repository (yocto-ktn) and the (customer) build repository (only if option `-u` is set) from the server and then parses the `repo.conf` file in the build repository.

The meta-layers with the specified revisions are then checked out to `yocto-exceet/layers` . The `meta-update` script needs to know the current build, but you usually don't need to set the `-b` option as the script gets the current build from an environment variable `BUILDDDIR` , that is set while running `init-env` .

To update the current build without using init-env you can run `meta-update` directly:

```
meta-update -u
```

If you only want to check out the meta-layers specified in `repo.conf` , maybe because you ran some manual `git checkout` commands in the layers and want to return to the state defined in `repo.conf` :

```
meta-update
```

Important

Please note that whenever you run `init-env` or `meta-update` and have local changes in one of the repositories, you can run into problems while the script tries to checkout a certain revision of the build repository or a meta layer. To resolve these problems, go to the repository and do one of the following steps, depending on your situation:

1. Discard your uncomittet changes if you do not need them anymore by running `git checkout -- .` or a similar command *or*
2. Stash your changes for later reuse, see: git stash *or*
3. Commit your changes and if necessary, push them to the remote. You might also want to update `repo.conf` afterwards.

Other Helpful Scripts

The `yocto-ktn/scripts` directory contains some more scripts, you might find helpful:

1. `meta-bump` updates your `repo.conf` . You can set a certain layer to a specific revision, or you can update all layers to the latest revision by running the script without any arguments.
2. `meta-status` prints information about the current state of the meta layers.
3. `init-remote2` initializes TFTP, NFS and a webserver on your local machine to use network boot on your target device and to be able to install packages on your target from a local pacakge server. It also can get its configuration from a file. For examples see the 'init-remote_*' files in 'conf' subdirectory your build directory.

Using Bitbake

To build a single package, an image for the target or a toolchain, Yocto uses the `bitbake` command. Before you can use it, you must set up your build environment with `init-env` . See Setting up and using the build environment.

Building a single recipe

To build a single recipe use:

```
bitbake <recipe-name>
```

To list all available recipes run:

```
bitbake -s
```

> **Demand of Resources**
>
> Please note, that building from scratch can take a long time (several hours!) and needs a lot of disk space and RAM! Especially when you build images with large libraries like Qt. To build as much as possible even when a recipe fails you can use the `-k` option for bitbake.

Building an image

The Kontron BSP provides three basic images:

- **image-ktn-minimal** is a minimal image which simply boots the hardware
- **image-ktn** is a basic console image with utilities for debugging, package-manager and SSH access.
- **image-ktn-qt** is a image with Qt5/EGLFS support with demo applications.

For information on which image is dedicated to your board, consult your platform and board documentation by selecting a platform/BSP in the top navigation dropdown menu.

To build, for example, the image-ktn type

```
bitbake image-ktn -k
```

and wait until Yocto finished its work. You can find the image files in the `tmp/deploy/images/<yourmachine>` directory of your build repository.

> **EULA**
>
> Depending on the platform, before building an image, you might have to read and accept the EULA document. Else the build will fail!
> See a freshly generated `local.conf` where to find the license documents and how to accept them for your board (e.g. set `ACCEPT_EULA_stm32mp-t1000-s-multi = "1"` in `local.conf`)

After you have built your image successfully, you now can go on to boot this image on your machine. For further guidance on this go the documentation of your specific platform.

Building the SDK for your image

After you have built your image, it is possible to build a SDK which fits your machine and image contents. For the Kontron Eval-Kits there are already precompiled SDKs. See Prebuilt BSP Releases for more info.

To create an installer for the toolchain of your board and image combination type:

```
bitbake <image-recipe> -c populate_sdk
```

For example:

```
bitbake image-ktn -c populate_sdk
```

For Qt5 development there is a special recipe which contains the tools needed for Qt5 development:

```
bitbake meta-toolchain-qt5
```

> **Two Different Toolchains/SDKs**
>
> `meta-toolchain-qt5` and `image-ktn-qt` are two toolchains with a little difference:
>
> - the `meta-toolchain-qt5` is one choice when you want to develop a GUI application based on the Qt framweork. This toolchain contains the biggest set of Qt extensions, but no additional libraries of your image. It contains the full set of Qt development tools.
> - the `image-*` toolchain fits exactly to your image configuration. This toolchain contains, besides the Qt extensions, all libraries included into your image. This toolchain is the best choice if you create an application which needs special libraries which are only part of your image.

After you created the toolchain-installer with Yocto, you can find it in the directory `<build-dir>/tmp/deploy/sdk` .

The installer is a shell-script, that can be executed like this:

```
sh ktn-glibc-x86_64-cortexa7t2hf-neon-vfpv4-image-ktn-qt-toolchain-
thud_1.3.1.sh
```

The default installation path is `/opt/kontron/` if not specified otherwise.

Modify the BSP

> **More about BSP-Customization**
>
> You can find more specific information and examples in the platform-specific BSP documentation.

> **Modifications and Collaboration**
>
> If you make any modifications to the BSP yourself and you want to use these changes in your final product, think about where to save your changes. If you work on a BSP together with KED engineers, you can use the KED GitLab server for your custom build and meta-layer repositories.

Local or temporary modifications

On the first initialization a `local.conf` file is created in the `conf` directory of your build. This file is usually not tracked by git and is meant to be used only for local or temporary changes. Please check `local.conf` and read the comments in the file to find out about some default options.

The default `local.conf` file includes the `sourcecode-version.conf` and `user.conf` file if available. `sourcecode-version.conf` is meant to contain the sourcecode version number for all sourcecode compiled (can be used as software release number). It should be kept in sync with `repo.conf` .

Furthermore the `user.conf` file is meant to hold user specific settings that may be different between different developers. One such example is the URL variable for the package server (PACKAGE_FEED_URIS).

Keep Modifications in the Right Place

If you have changes in your `local.conf` that should not stay local, but need to be set as default for everyone who uses the build, then find a way to move these changes to the correct file. Some popular places in the meta layers are:

1. The image recipe in `recipes-core` for image-specific settings
2. The distro config in `conf/distro` for distro-specific settings
3. The machine config in `conf/machine` for machine-specific settings
4. Recipe of some package for package-specific settings

Contribute to KED layers

If you need to work around a problem in the KED layers by adding code to your own layer, think about whether you can fix the root of the problem in the KED layer and create a "Merge Request" with the changes for the KED repository (e.g. meta-ktn).

Create your own layers

If you want to make modifications to the BSP, we suggest to create your own layers to keep your modifications reproducible and to separate them from other layers. For customer boards Kontron uses a `build-<CUSTOMER>` repository and a `meta-<CUSTOMER>` layer to keep the modifications for special customer boards.

The `build-<CUSTOMER>` repository describes which layers in which version are required to build the product. It works just like the KED build repositories, which you can use as a blueprint for your own build repository.

The `meta-<CUSTOMER>` layer holds all adaptions for the custom hardware and software. This can be additional recipes or adaptions to some recipes, separate images, configurations and so on.

Also see the Yocto documentation for creating layers.

Using devtool to work on source code

To work on the source code of any package it is most convenient to use the `devtool` utility. As an example we will show how to modify the kernel code.

> **Devtool**
>
> For more information about the substantial `devtool`, please visit the Yocto Manual.

Example with linux-stm32mp

To start working on the `linux-stm32mp` code:

```
devtool modify linux-stm32mp
```

A separate workspace layer will be created and the kernel source tree will be extracted there. Do your code changes and run a build with:

```
bitbake linux-stm32mp
```

Test your changes and create patches if necessary. To reset to the previous state and build without the changes in the workspace run:

```
devtool reset linux-stm32mp
```

Tools, Apps and Resources

Resources

Prebuilt BSP Releases

You can find prebuilt BSP releases for some boards in https://files.kontron-electronics.de. Select the subdirectory for your platform to find images, sdk, open source sourcecode and license texts for your board. Check this location for new prebuilt software for your Eval-Kit.

Tools, Apps and Demos

The Kontron BSPs include or provide several demo applications and tools, which can be helpful to get started and to configure your hardware.

ptool

ptool (formerly `production-tool.sh`) is a set of scripts used to execute tasks for production purposes, such as flashing firmware to memory, setting boot configurations or running tests. Depending on the image configuration ptool might be already installed, but not all tasks might be executable.

To view a list of available tasks, run:

```
ptool -h
```

To run a specific task:

```
ptool <task_name>
```

e.g to write a rootfs image from SD-card to eMMC:

```
ptool flash_emmc
```

Some more task names are `flash_ubi` to flash the rootfs image to UBI NAND flash and `flash_bl` to flash the bootloader.

C-app-demo

A simple Hello-World application in C.

kontron-demo (QML)

A QML-based demo application, featuring a simple touch UI.

imagegestures and animatedtiles (Qt Widgets)

Two modified Qt examples to show performance of Qt-Widgets-based applications (also usable without GPU, e.g. on i.MX6UL-based HW).

Web viewer with virtual keyboard

The package `webengine-vk` contains a simple web browser with integrated onscreen touch keyboard. After installation with opkg (see Package Management) the application will autostart after booting. You can load a specific website or rotate the screen by running the application by launching the application manually from the command line:

```
> /opt/webengine-vk/webengine-vk rot 180 http://www.kontron-electronics.de
```

Please note that the web viewer is only capable to display a single page at the same time. No tab browsing is possible.

Using the Qt Cross Toolchain and QtCreator

By using a cross toolchain on your development computer, you can easily create Qt5 or bare C/C++ applications to run on your Kontron hardware.

Setting up the SDK

To generate a toolchain/SDK that matches the target image, you can run the `populate_sdk` task for your image. For example:

```
bitbake image-ktn-qt -c populate_sdk
```

There might also be a SDK installer available for your target platform on the. Kontron server at https://files.kontron-electronics.de.

If you created the toolchain-installer with Yocto, you can find it in the directory `<build-dir>/tmp/deploy/sdk` .

The installer is a shell script, that can be executed like this:

```
./ktn-glibc-x86_64-image-ktn-qt-aarch64-toolchain-ktn-zeus_3.0.0-alpha3.sh
```

The default installation path is `/opt/kontron/` if not specified otherwise.

Installing Qt/QtCreator

Check the download page at http://download.qt.io/official_releases/qtcreator for the latest release of QtCreator and download the `qt-creator-opensource-linux-x86_64-X.X.X.run` file.

Install QtCreator by double-clicking the `*.run` file in the file-manager or by running:

```
chmod +x qt-creator-opensource-linux-x86_64-4.11.2.run
./qt-creator-opensource-linux-x86_64-4.11.2.run
```

Alternatively you might also want to consider installing the full Qt environment for your desktop, including sources, tools and QtCreator. The advantage is, that apart from the Yocto-based target toolchain, you also have a toolchain available for your desktop environment. This enables you to switch between deploying to the target and deploying to your desktop machine for testing.

The online installer for the latest OpenSource edition of Qt can be found here.

Configuring and using QtCreator

For cross development using the Yocto SDK/toolchain, you need to pass the build environment settings to QtCreator. There are two ways to do this.

Using the helper script (only available in i.MX BSP from version 3.0 on)

The latest versions of the Kontron Yocto Qt SDKs provide a script, that creates the "Kit" with all the settings within QtCreator. It also copies the build environment variables to the Kit, so you can easily switch between different SDKs from within QtCreator and you don't need to source the build environment before starting QtCreator.

The script can be found in the SDK directory which defaults to `/opt/kontron/<MACHINE>/<VERSION>` .

For example:

```
./qtcreator-setup-helper-armv7vet2hf-neon-ktnsdk-linux
Please enter the path of your QtCreator installation or
press enter for default (~/Qt/Tools/QtCreator):
Using default path ~/Qt/Tools/QtCreator
Created toolchain/compiler with id
'ProjectExplorer.ToolChain.Gcc:ktn-zeus-kontron-mx6ul'
Created debugger with id 'ktn-zeus-kontron-mx6ul'
Created Qt version with id 'ktn-zeus-kontron-mx6ul'
Created kit with id 'ktn-zeus-kontron-mx6ul'
```

Manual Setup

Running QtCreator within the build environment

To set the build environment for QtCreator manually you can edit the start script at `<qtcreator-install-dir>/bin/qtcreator.sh` (e.g. `~/Qt/Tools/QtCreator/bin/qtcreator.sh`).

Add the following line at the very beginning of the file (before `#!/bin/sh`) and change the SDK path and script name accordingly:

```
source /path/to/sdk/environment-setup-cortexa9hf-vfp-neon-ktn-
linux-gnueabi
```

This will source the build environment upon starting QtCreator. Please ensure, that you are **always starting QtCreator through the qtcreator.sh script**, and not directly by executing the binary.

You can modify the Ubuntu Launcher icon to make this easier. On Ubuntu 18.04 this should work:

```
gedit ~/.local/share/applications/org.qt-project.qtcreator.desktop
```

The modification is to add `bash -c` and an additional `.sh` to the content of the `Exec` variable, so that after clicking on the symbol the script is called:

```
Exec="bash -c /home/Qt/Tools/QtCreator/bin/qtcreator.sh" %F
```

Configuring the SDK Kit

Kits can be modified in the *Build&Run* view (`Tools -> Options -> Build & Run`). Here you can configure your kit with:

- its name (choose what you want)
- the device type (*Generic Linux Device* for Kontron devices)
- the device to be used (see Adding your own device)
- the sysroot - headers and libraries for the Yocto firmware (/sysroots/)
- cross-compiler (/sysroots/x86*/usr/bin/arm-ktn-linux-gnueabi/arm-*-g++)
- cross-debugger (/sysroots/x86*/usr/bin/arm-ktn-linux-gnueabi/arm-*-gdb)
- Qt version (/sysroots/x86*/usr/bin/qt5/qmake)

The configuration entries for compiler, debugger and Qt version have several tabs to enter the matching configuration settings. In the kits view you can only select already predefined settings.

Deploying to the target

Adding your own device

To be able to use the deploy and debugging features within QtCreator with your own device you can modify an existing configuration or create a new one.

If you want to create a new configuration in QtCreator or modify an existing one go to `Tools -> Options -> Devices` and add your device by clicking `Add ...` . Choose `Generic Linux Device` from the drop down menu. Set the appropriate values for the device name, IP address and the username and password (if any). The connection can be tested with the `Test` button in the devices tab.

Deploy to the target device

QtCreator uses sftp and ssh to copy the files and run the application remotely.

The deployment of your files is configured in your qmake project file by setting the `INSTALLS` variable. The Kontron demo programs contain some simple deployment rules. For more information see the Qt/qmake documentation.

Copy files manually to the target

Using the protocols sftp and scp files can be copied to the target. If the target has the IP address `192.168.0.10` the content of the target can be shown by using the URL `sftp://root@192.168.0.10/` in a browser window.

To copy files via command line the program scp can be used. You can also start a remote shell on the target via ssh:

```
ssh root@192.168.0.10
root@192.168.0.10:~
```

Debbuging

If gdbserver is running on the target (default in image-exceet/image-ktn) you should be able to use the remote debugging features in QtCreator.

If you want to debug QML based applicaitons ensure, that (QML-)debugging is enabled in the project settings. If you are trying to debug a QML-app and you get an error message "Invalid Signal", then try to skip the message by clicking 'OK' and the use the button with the green arrow to continue debugging.

Further hints for debugging

Disable optimization

Debugging an optimized binary might be difficult because the compiler may reorder or optimize away some code. To disable optimization for debugging

purposes you can set some QMAKE variables in your project file to disable optimization. But be aware that this might have side effects!

```
QMAKE_CXXFLAGS_DEBUG += -O0
QMAKE_CFLAGS_DEBUG += -O0
```

Rejected loading of shared libraries

Loading of shared libraries is sometimes rejected by gdb due to 'insecure path settings'. Put these gdb commands into your QtCreator configuration for gdb to disable secure path setting.

```
set auto-load safe-path /
```

You can configure additional start commands for gdb through `Tools -> Options -> Debugger -> GDB`.

Using gdb-multarch (applies to Yocto Morty)

Qt Creator uses so called *pretty printers* to provide a easy interface to basic Qt classes like e.g. Qt strings. These pretty printers are implemented with the help of python functions in gdb. The cross toolchain for the devices may lack some python libraries Qt creator needs for its pretty printers. To circumvent this use the `gdb-mulitarch` debugger of your development machine for remote debugging (already default for morty). You can install `gdb-multiarch` on your development host by running `sudo apt install gdb-multiarch`.

Qt Environment

Qt offers several options concerning framebuffer, eglfs, input devices, etc. that can be set via environment variables. Please use this page as a reference for the available options: Qt for Embedded Linux.

Also check the specific documentation for your platform/BSP, as it may contain further details for setting up Qt for your device.

QML Software Rendering

To be able to use QML/QtQuick applications on SoCs without GPU, it is possible to select the included software renderer. This enables you to use QML/QtQuick for example on our SoMs with i.MX6UL/ULL. Some features such as shaders, etc. won't be available and you might experience performance issues, depending on how your application is designed.

To use the software backend, set `QT_QUICK_BACKEND=software` in your environment.

Creating a Yocto recipe for a Qt application

See the Yocto-Dev-Manual for more information on how to write your own recipe.

You can also consult the meta-ktn layer for examples.

Contribute

FOSS

Most parts of our software and documentation are **open-source**. We support the idea behind **FOSS** and actively contribute to projects such as **Linux, U-Boot and Yocto/OE**.

If you work with our hard- and software or with the documentation, we would like to encourage you to **contribute** by **reporting issues** or **sending merge requests**.

Repositories

For a list of public repositories, please see **this page on the GitLab server**. You can also have a look at the Git Server Overview in the docs.

Reporting Issues

You can use the GitLab issue trackers for the specific projects to report any problems you encountered and you think should be known to KED engineers.

Contributing code changes

If you have forked one of the KED repositories and you have implemented changes that might be relevant or helpful for others, please send a "merge request" (MR) via GitLab. We will review your changes and merge them if appropriate.

Upstreaming

We try to work with "mainline"-components wherever possible. We also try to upstream support for our (demo-/eval-) boards and modules to Linux and U-Boot eventually. If you work with one of our boards, that is not yet available upstream and you are interested in upstream support, feel free to contact us at support@kontron-electronics.de for more information and how to work it out.

Issue Tracker

Please have a look at the issue tracker on our GitLab server for known bugs and issues affecting the BSP and also to report any issues you might encounter.

Other Known Issues

NXP i.MX Chip Erratas

- IMX6 UltraLight
- IMX6 ULL
- IMX6 Solo/DualLight
- IMX6 Dual/Quad
- IMX8M-Mini

Library version mismatches while debugging

Occurence

While cross-debugging (e.g. with QtCreator) you might note messages like these:

```
.dynamic section for "/opt/exceet/mx6sexceet/sysroots/cortexa9hf-
vfp-neon-exceet-linux-gnueabi/usr/lib/libQt5Qml.so.5" is not at
the expected address (wrong library or version mismatch?)
```

Reasons

The root cause of this is often that the library version in the toolchain, that is used differs from the one on the target.
Sometimes it can happen, that you use a new image and although nothing has changed in the library since the previous build of the toolchain, these

messages occur. This is because sometimes paths relative to the build directory on the build host are compiled into the binaries. If this information changes, e.g. because the new image was build on an other machine, it might happen that the addresses in the library files are shifted and though the content of the library is essentially the same, its structure has slightly changed, what produces the messages in the debug log.

Measures

This problem has been approached in recent versions of Yocto by trying to strip all the absolute paths from the target binaries.
Check what caused the changes in the library and decide whether to build/ install a new toolchain or not.

VMware Problems

No log-in possible after booting

Suspend your image. Then start your image again, click immediately into the window and press the left shift key. Now the boot process will stop and the Ubuntu boot menu is presented. Choose here to boot in recovery mode.
After the recovery mode is started, choose to resume normal booting. After a while you will be able to login into Ubuntu.
To fix the problem permanently open the file '/etc/gdm3/custom.conf' with the editor of your choice (sudo priviliges needed) and uncomment the line saying '#WaylandEnable=false'. With that fix you should be able to login again after normal booting.
(Copied from https://askubuntu.com/questions/1149957/unable-to-login-to-account-in-ubuntu-18-04-vmware-workstation-15-after-update)