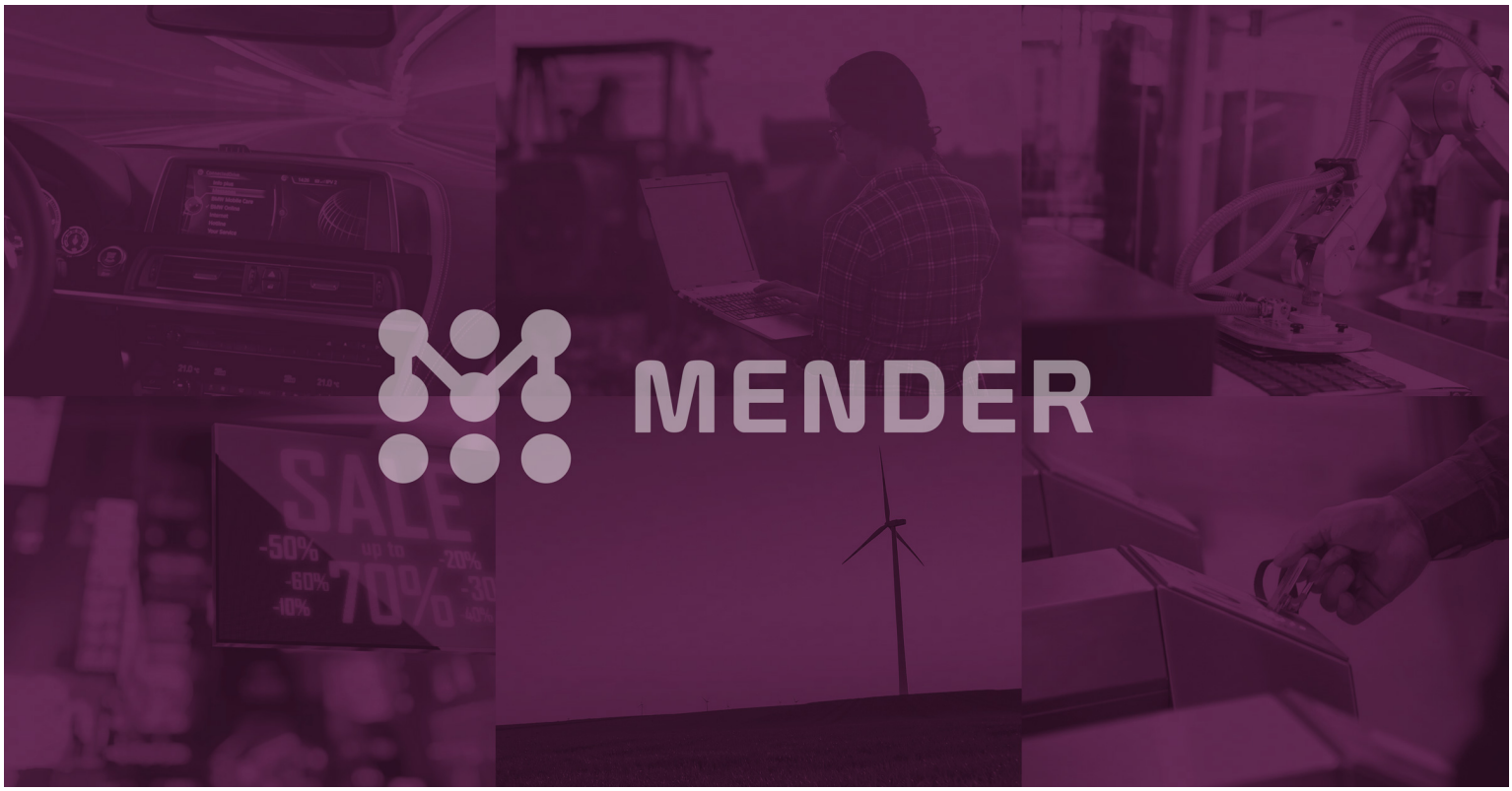

The Yocto Project Software Development Kit (SDK) QuickStart Guide

by Mender.io

The Yocto Project Software Development Kit (SDK) QuickStart

Welcome to the Yocto Project's Software Development Kit (SDK) QuickStart Guide by Mender.io. Mender.io is an open source project to deliver over-the-air software updates for Yocto-based devices and we are providing this Guide to get you started on the Yocto SDK – this QuickStart is not specific to any Mender software or procedures.

This document introduces the Yocto Project and SDK development environment and allows you the opportunity to quickly see how to add new functionality as well as modify existing functionality to an existing image that is already able to boot and run on targeted embedded hardware.

The following procedures will simply allow you, the application developer, to see how to set up and then develop using the Yocto Project SDK. After working through the examples in this QuickStart, you should have an understanding of the Yocto Project and the SDK development environment, be able to quickly add a new application to an existing image, and be able to modify an existing application used by an image.

For further reading on the Yocto Project, see the following two publications:

- The Yocto Project QuickStart
- The Yocto Project Software Development Kit (SDK) Developer's Guide

Introduction

The Yocto Project is an open-source collaboration project focused on embedded Linux developers. Part of the Yocto Project development environment is an SDK development environment. Using the Yocto Project's SDK development environment, you can quickly develop, test, and deploy applications as part of embedded images running on actual hardware all without having to work within the full Yocto Project development environment. In other words, you can take advantage of the popular Yocto Project environment without having to spend time ramping up on the full-blown environment.

If you want to learn more about the Yocto Project, visit their website at http://www.yoctoproject.org and take some time to examine the documentation set.

Understanding and Preparing the Development Environment

The development environment consists of a "build system" running a Linux distribution, which you use to host the SDK and perform your development work. Although the build system does not require the Yocto Project be installed, it must be running a distribution that supports the Yocto Project. See the Yocto Project Reference Manual for the list of supported Linux distributions.

Following are the requirements for your development environment:

- **Existing Image**: The image that currently runs on your target hardware. This is the image to which you are adding new functionality or modifying existing functionality. For this QuickStart, we are going to use an image that can run on the Quick EMUlator (QEMU).
- **SDK**: The installed Software Development Kit. It is recommended that you have the extensible SDK installed so that you can easily add new functionality and modify existing functionality. The examples in this QuickStart use the extensible SDK.
- **Initialized SDK Environment**: The properly initialized SDK development environment, which is set up by running the SDK environment setup script.
- **Source Code**: Existing source files for any new functionality or for any existing functionality you would like to add or modify, respectively. For the QuickStart, we will add a "helloworld" module as new functionality and we will modify the existing multiple device administrator ("mdadm") module, which is a core module included as part of the SDK that you install.

Use these steps to set up your SDK development environment for the examples in this QuickStart:

1. Open a browser and locate images that can be run using QEMU:

   http://downloads.yoctoproject.org/releases/yocto/yocto-2.1/machines/qemu/qemux86-64

2. Download and save this image:

   `core-image-sato-dev-qemux86-64.ext4`

3. Use the browser to locate the SDKs at this URL:

   http://downloads.yoctoproject.org/releases/yocto/yocto-2.1/toolchain/x86_64

4. It is very important that when you choose the SDK, you choose the one appropriate for the hardware or for QEMU. In this case, you need the extensible SDK that maps to the image we are going to eventually run on QEMU. Download and save this extensible SDK:

   poky-glibc-x86_64-core-image-sato-core2-64-toolchain-ext-2.1.sh

   For more details on selecting and installing an SDK, see the "Installing the SDK" section in the Yocto Project Software Development Kit (SDK) Developer's Guide.

5. Make the installer executable:

```
$ cd ~/Downloads
$ chmod +x poky-glibc-x86_64-core-image-sato-core2-64-toolchain-ext-2.1.sh
```

6. Install the SDK by running the script and accepting the defaults:

```
$ ./poky-glibc-x86_64-core-image-sato-core2-64-toolchain-ext-2.1.sh
Poky (Yocto Project Reference Distro) Extensible SDK installer version 2.1
==========================================
Enter target directory for SDK (default: ~/poky_sdk):
You are about to install the SDK to "/home/scottrif/poky_sdk". Proceed[Y/n]? y
Extracting SDK.....……..done
Setting it up…
Extracting buildtools…
Preparing build system…
done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the
environment setup script e.g. $ . /home/scottrif/poky_sdk/environment-setup-core2-64-
poky-linux
```

7. Run the SDK Development Environment initialization script:

```
$. $HOME/poky_sdk/environment-setup-core2-64-poky-linux
SDK environment now set up; additionally you may now run devtool to perform
```

```
development tasks.
Run devtool --help for further details.
```

8. Ensure your new Source Code is available. The only requirement for the source code is that it either exists locally or can be accessed through an URL for downloading. This example assumes the source code exits here:

```
$HOME/mysource/helloworld
```

NOTE: Several methods exist by which you can gain access to your source files. See the "Use devtool add to Add an Application" section in the Yocto Project Software Development Kit (SDK) Developer's Guide for more information.

For this example, the source files must consist of the following files: `hello.c`, `Makefile.am`, and `configure.ac`:

The `helloworld/hello.c` file is as follows:

```
#include <stdio.h>

main()
   {
      printf("Hello World:\n");
   }
```

The `helloworld/Makefile.am` file is as follows:

```
bin_PROGRAMS = hello
hello_SOURCES = hello.c
```

The `helloworld/configure.ac` file is as follows:

```
AC_INIT(hello,0.1)
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CC
AC_PROG_INSTALL
AC_OUTPUT(Makefile)
```

Adding New Functionality

Now that all the pieces are in place, you can use `devtool`, which is a robust tool that facilitates development in the SDK environment, to develop and test the new "helloworld" module. The result will be a built object that can be deployed to QEMU and tested from the running image.

Follow these steps to build, deploy, and test the new "helloworld" module:

1. Use `devtool add` to locate and set up the work environment named "workspace" for development. The command automatically creates the recipe needed to build the module:

```
$ cd $HOME/poky_sdk
$ devtool add helloworld $HOME/mysource/helloworld
NOTE: Recipe /home/scottrif/poky_sdk/workspace/recipes/helloworld/helloworld.bb has been
automatically created; further editing may be required to make it fully functional.
```

2. Use the `devtool build` command to run the recipe using the Yocto Project build engine, which is part of the installed SDK, and create the build output for your new module:

```
$ devtool build helloworld
Loading cache: 100% |################################################| ETA:  00:00:00
Loaded 1302 entries from dependency cache.
Parsing recipes: 100% |##############################################| Time: 00:00:00
Parsing of 872 .bb files complete (871 cached, 1 parsed). 1302 targets, 48 skipped, 0
masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
NOTE: Preparing RunQueue
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
helloworld-0.1-r0 do_compile: NOTE: helloworld: compiling from external source tree
/home/scottrif/mysource/helloworld
NOTE: Tasks Summary: Attempted 342 tasks of which 336 didn't need to be rerun and all
succeeded.
```

3. For this example, make sure the image you downloaded earlier is in the `poky_sdk` directory where the SDK was installed and then use the `devtool runqemu` command below to start QEMU and boot the image.

```
$ cp $HOME/Downloads/core-image-sato-dev-qemux86-64.ext4 $HOME/poky_sdk
$ devtool runqemu core-image-sato-dev-qemux86-64.ext4
```

If your system does not have graphical support, use the “nographic” option as follows:

```
$ devtool runqemu nographic core-image-sato-dev-qemux86-64.ext4
```

NOTE: You can provide a full pathname to the image if you do not want to copy it to the local directory from which you are running `devtool`.

4. Because the previous step ties up the terminal, you need to open a new terminal window, get into the SDK development area, and run the SDK initialization script to continue:

```
$ cd $HOME/poky_sdk
$ . $HOME/poky_sdk/environment-setup-core2-64-poky-linux
```

5. Use the `devtool deploy-target` command to deploy the “helloworld” output you built earlier to QEMU so that you can test the module.

```
$ devtool deploy-target helloworld root@192.168.7.2
```

NOTE: If you get an error message indicating the script failed to copy to root@192.168.7.2, you can run the command with the “-c” option to turn off key checking:

```
$ devtool deploy-target -c helloworld root@192.168.7.2
```

6. Run the “hello” command at the terminal where QEMU was booted:

```
sh-4.3# hello
Hello World:
sh-4.3#
```

You will see “Hello World” returned as terminal output. At this point, you have added the new functionality and proved that it works.

7. Make sure to upload the sources into a repository (e.g. Git) such that the files could be fetched by other people. This is typical practice once any new module has been created and tested and is ready for use by other developers.

8. At this point you have put the source code in a more permanent area where it can be fetched by someone using the full-blown YP development process. You need to know where the recipe is:

```
$HOME/poky_sdk/workspace/recipes/helloworld
```

Modifying Existing Functionality

Another common scenario is to modify code that already has an existing recipe in place. This is different that adding brand new functionality as was demonstrated in the previous section.

To help demonstrate this concept, we will use a more "real-world" example: the existing multiple device administrator ("mdadm") module, which is a core module included as part of the installed SDK. The steps in this section will modify "mdadm" to change the reported version, which is 3.4.

Follow these steps to modify "mdadm":

1. Use the `devtool modify` command to locate, extract, and prepare the "mdadm" files. The command locates the source based on information in the existing recipe, unpacks the source into the "workspace", applies any existing patches, and parses all related recipes:

```
$ devtool modify mdadm
Parsing recipes..done.
NOTE: Fetching mdadm…
NOTE: Unpacking…
NOTE: Patching…
NOTE: Adding local source files to srctree…
NOTE: Source tree extracted to /home/scottrif/poky_sdk/workspace/sources/mdadm
NOTE: Using source tree as build directory since recipe inherits autotools-brokensep
NOTE: Recipe mdadm now set up to build from
/home/scottrif/poky_sdk/workspace/sources/mdadm.
```

2. In the source location (`$HOME/poky_sdk/workspace/sources/mdadm`), use an editor to edit the "ReadMe.c" file. Change the line that specifies the version as "3.4" to be "3.4-modified".

```
#define VERSION "3.4-modified"
```

3. Rebuild "mdadm" using the `devtool build` command:

```
$ devtool build mdadm
Parsing recipes: 100%
|##################################################################################################
########################| ETA:  00:00:00
Parsing of 872 .bb files complete (0 cached, 872 parsed). 1302 targets, 48 skipped, 0
masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
NOTE: Preparing RunQueue
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
mdadm-3.4-r0 do_compile: NOTE: mdadm: compiling from external source tree
/home/scottrif/poky_sdk/workspace/sources/mdadm
NOTE: Tasks Summary: Attempted 347 tasks of which 336 didn't need to be rerun and all
succeeded.
```

4. QEMU is still assumed to be running in the shell from the previous section. Use the `devtool deploy-target` command to deploy the built "mdadm" module to the already running image:

```
$ devtool deploy-target mdadm root@192.168.7.2
NOTE: Successfully deployed /home/scottrif/poky_sdk/tmp/work/core2-64-poky-
linux/mdadm/3.4-r0/image
```

5. In the shell running QEMU, verify your changes by running "mdadm" and providing the "-V" option to return the version:

```
sh-4.3# mdadm -V
mdadm - v 3.4-modified 21 June 2016
sh-4.3#
```

NOTE

If you try to use the previous command to see the original version of "mdadm", you will get a message indicating the command is not found. "mdadm" is not in the image by default.

6. At this point, you have modified the source code of an existing module all within the "workspace". You will need to commit those changes (i.e Git commit).
Use the following two commands to see the changes that need to be staged and committed:

```
cd $HOME/poky_sdk/workspace/sources/mdadm
$ git status
```

Use these two commands to stage and commit the changed file:

```
$ git add ReadMe.c
$ git commit -m 'ReadMe.c-Updated-the-version-string'
```

7. Turn the commit you have made into a patch:

```
$ devtool update-recipe mdadm
Parsing recipes..done.
NOTE: Updating file run-ptest
NOTE: Adding new patch 0001-ReadMe.c-Updated-the-version-string.patch
NOTE: Updating recipe mdadm_3.4.bb
```

The patch is located here:

```
$HOME/poky_sdk/layers/build/meta/recipes-extended/mdadm/mdadm/0001-ReadMe.c-Updated-the-version-string.patch
```

Summary

If you successfully worked through the procedures in this Quick Start, you should now understand the following:

- The high-level purpose and role of the Yocto Project SDK.
- What hardware and software pieces are needed in order for you to use the SDK to develop functionality outside of the main Yocto Project development environment.
- How to set up your build system so that you can use the SDK.
- Where to locate images and SDKs that allow you to develop functionality.
- How to create and test new functionality that you can integrate into an existing image.
- How to update and test existing functionality that you can re-integrate into an existing image.

For further understanding of the Yocto Project development environment, we encourage you to examine the documentation on the Yocto Project main website at http://www.yoctoproject.org.