SDK for OpenSTLinux distribution

Stable: 07.01.2021 - 12:42 / Revision: 07.01.2021 - 12:38

A quality version of this page, approved on *7 January 2021*, was based off this revision.

Contents

1 Article purpose

This article aims to give **general information** about the software development kit (SDK) for the OpenSTLinux distribution.

Information

To install and use efficiently the last release of the OpenSTLinux SDK, please read the Developer Package article relative to the Series of your STM32 microprocessor: Category:Developer Package

2 Introduction

The **software development kit (SDK)** for the OpenSTLinux distribution is a customization of the Yocto SDK[1], which provides a **stand-alone cross-development toolchain** and libraries tailored to the contents of a specific image. The OpenSTLinux SDK is part of the STM32MPU Embedded Software Developer Package.

The **SDK might be generated**, through the STM32MPU Embedded Software Distribution Package, during the compilation of a software release, which guarantees the alignment of this SDK with the software images (binaries) built for the Starter Package of the STM32MPU Embedded Software: see SDK development cycle model.

It provides a more "traditional" toolchain experience than the full Yocto project (OpenEmbedded) development environment available through the Distribution Package of the STM32MPU Embedded Software.

It simplifies the workflow for application developers: it has no dependency on the Yocto project used for its generation (Distribution Package), and can be installed on any host machine. Note that many SDKs can coexist on the same host machine.

3 Why use the SDK, and how?

The OpenSTLinux SDK gives developers an efficient development cycle (compilation, deployment on target, and debug).

Using this SDK, developers take advantage of the Yocto project development environment (to quickly develop, deploy and test applications, or any other piece of software, as part of images running hardware), without having to understand all the Yocto project mechanisms that might seem somewhat complex.

3.1 SDK development cycle model

A developer can install the SDK on a machine (host PC), and use it to develop within any piece of software (for example, an application, kernel drivers or kernel modules).

Basically, the developer has to:

- get the software images (binaries) of the software release associated with the SDK (see Starter Package)
- install the SDK for the targeted hardware (see SDK installation)
- run the SDK environment setup script (see SDK startup)
- develop and test the piece of software

When the development is finished (the source code is ready to be shared with other developers), it should be integrated into the whole software. For this, the Distribution Package must be used.

Through the Distribution Package, new images (binaries) and a new SDK are generated (see How to create an SDK for OpenSTLinux distribution).

4 SDK content

The OpenSTLinux SDK is based on the **standard** Yocto project SDK.

A standard SDK consists of the following:

- a cross-development toolchain: this toolchain contains a compiler, linker, debugger, and various miscellaneous tools
- libraries, headers, and symbols (target and native sysroots): the libraries, headers, and symbols are specific to the image (that is, they match the image)
- an environment setup script: this *.sh file, once run, sets up the cross-development environment by defining variables and preparing it for SDK use

4.1 Cross-development toolchain

The cross-development toolchain consists of a cross-compiler, a cross-linker and a cross-debugger that are used to:

- develop user-space applications for targeted hardware
- modify a software component that already exists in the images, and that is delivered as source code in the Developer Package (for example the Linux kernel or U-Boot)

This cross-development toolchain is created by running a toolchain installer script (see SDK installation).

It works with a matching target sysroot (see below).

4.2 Native and target sysroots

The native and target sysroots contain the required headers and libraries for generating binaries that run on the target architecture.

The target sysroot is based on the target root file system image that is built through the Distribution Package of the STM32MPU Embedded Software and uses the same metadata configuration as that used to build the cross-toolchain.

For any software baseline, this process guarantees the alignment between:

- the content (source code) of the Distribution Package
- the target root file system image (binary) of the Starter Package
- the target sysroot (headers and libraries) of the Developer Package
- the configuration of the cross-toolchain of the Developer Package

5 SDK installation

The OpenSTLinux SDK is installed on the host development machine by running the *.sh installation script.

The tarball file (*SDK-[...].tar.xz*) that contains this script is named as follows: **SDK-<host machine>-<version>.tar.xz** where:

<host machine>	Host machine on which the SDK is installed: • x86_64 (only 64-bit host machines are supported)
<version>	Software release version; example: • openstlinux-5.4-dunfell-mp1-20-11-12

Example:

- en.SDK-x86_64-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz

The steps for the OpenSTLinux SDK installation, are:

- Download, on the host machine, the SDK tarball file (*SDK-[...].tar.xz*)
- Decompress the tarball file

```
$ tar xvf en.SDK-[...].tar.xz
```

- The **installation script** is named:

  <image>-<distro>-<machine>-<host machine>-toolchain-<Yocto release>-<version>.sh

  where:

<image>	Image name; example: • st-image-weston
<distro>	Distribution name; example: • openstlinux-weston
<machine>	Machine name; example: • stm32mp1
<host machine>	Host machine on which the SDK is installed: • x86_64 (only 64-bit host machines are supported)
<Yocto release>	Release number of the Yocto Project; example: • 3.1 (aka dunfell)
<version>	Software release version; example: • openstlinux-5.4-dunfell-mp1-20-11-12

  Example:

  - st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-3.1-openstlinux-5.4-dunfell-mp1-20-11-12.sh

- If necessary, change the permissions on the installation script so that it is executable:

```
$ chmod +x <image>-<distro>-<machine>-<host machine>-toolchain-<Yocto release>-
<version>.sh
```

- The SDK is self-contained and by default is installed into */opt/st/<machine>/<Yocto release>-<version>*

  Example:

  - /opt/st/stm32mp1/3.1-openstlinux-5.4-dunfell-mp1-20-11-12

  However, running the installation script with the *-d* option allows an installation directory to be chosen

  Check that the write permissions in the installation directory (either the default one, or the customized one) are granted

Information

Recommendation: for an STM32MPU Embedded Software release, install the software image from the Starter Package, the SDK and the source codes from the Developer Package in the same top directory; indeed, these packages are linked

- Run the installation script

```
$ ./<image>-<distro>-<machine>-<host machine>-toolchain-<Yocto release>-<version>.sh
```

Example (with an installation directory */local/SDK/<Yocto release>-<version>* different from the default one)

```
$ ./st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-2.6-openstlinux-20-02-19.
sh -d /local/SDK/2.6-openstlinux-20-02-19
ST OpenSTLinux - Weston - (A Yocto Project Based Distro) SDK installer version 2.6-
openstlinux-20-02-19
==========================================================================================
============================
You are about to install the SDK to "/local/SDK/2.6-openstlinux-20-02-19". Proceed[Y/n]? Y
Extracting SDK.....................................................................
.........................done
Setting it up...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.
 $ . /local/SDK/2.6-openstlinux-20-02-19/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-
linux-gnueabi
```

The OpenSTLinux SDK install is now complete.

Refer to Standard SDK directory structure for details of the resulting directory structure of the installed SDK.

6 SDK startup

To use an installed SDK, its environment setup script must be run.

This setup script is located in the SDK installation directory (per default, */opt/st/<machine>/<Yocto release>-<version>*).

It must be run once in each new working terminal.

This **environment setup script** is named:

environment-setup-<target>-<distro>-linux-gnueabi

Where:

<target>	Target architecture for cross-toolchain; example: • cortexa7t2hf-neon-vfpv4
<distro>	Distribution name; example: • openstlinux_weston

Example

- environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi

- Run the environment setup script

```
 $ source <SDK installation directory path>/environment-setup-<target>-<distro>-linux-gnueabi
```

Example: here, the SDK installation directory (*/local/SDK/<Yocto release>-<version>*) is different from the default one

```
 $ source /local/SDK/3.1-openstlinux-20-11-12/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
```

- Many environment variables are then defined:

```
SDKTARGETSYSROOT - the path to the sysroot used for cross-compilation
PKG_CONFIG_PATH - the path to the target pkg-config files
CONFIG_SITE - a GNU autoconf site file preconfigured for the target
CC - the minimal command and arguments to run the C compiler
CXX - the minimal command and arguments to run the C++ compiler
CPP - the minimal command and arguments to run the C preprocessor
AS - the minimal command and arguments to run the assembler
LD - the minimal command and arguments to run the linker
GDB - the minimal command and arguments to run the GNU Debugger
STRIP - the minimal command and arguments to run 'strip', which strips symbols
RANLIB - the minimal command and arguments to run 'ranlib'
OBJCOPY - the minimal command and arguments to run 'objcopy'
OBJDUMP - the minimal command and arguments to run 'objdump'
AR - the minimal command and arguments to run 'ar'
NM - the minimal command and arguments to run 'nm'
TARGET_PREFIX - the toolchain binary prefix for the target tools
```

```
      CROSS_COMPILE - the toolchain binary prefix for the target tools
      CONFIGURE_FLAGS - the minimal arguments for GNU configure
      CFLAGS - suggested C flags
      CXXFLAGS - suggested C++ flags
      LDFLAGS - suggested linker flags when you use CC to link
      CPPFLAGS - suggested preprocessor flags
```

The OpenSTLinux SDK is started.

7 References

- Yocto Project Application Development and Extensible Software Development Kit (eSDK)

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

Linux® is a registered trademark of Linus Torvalds.

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

also known as

GNU dedugger, a portable debugger that runs on many Unix-like systems