Yocto Project Application Developer's Guide

Jessica Zhang, Intel Corporation **<jessica.zhang@intel.com>**

by Jessica Zhang

Note

For the latest version of this manual associated with this Yocto Project release, see the Yocto Project Application Developer's Guide [http://www.yoctoproject.org/docs/1.9/adt-manual/adt-manual.html] from the Yocto Project website.

Table of Contents

Chapter 1. Introduction

Welcome to the Yocto Project Application Developer's Guide. This manual provides information that lets you begin developing applications using the Yocto Project.

The Yocto Project provides an application development environment based on an Application Development Toolkit (ADT) and the availability of stand-alone cross-development toolchains and other tools. This manual describes the ADT and how you can configure and install it, how to access and use the cross-development toolchains, how to customize the development packages installation, how to use command-line development for both Autotools-based and Makefile-based projects, and an introduction to the Eclipse™ IDE Yocto Plug-in.

Note

The ADT is distribution-neutral and does not require the Yocto Project reference distribution, which is called Poky. This manual, however, uses examples that use the Poky distribution.

Chapter 2. The Application Development Toolkit (ADT)

Part of the Yocto Project development solution is an Application Development Toolkit (ADT). The ADT provides you with a custom-built, cross-development platform suited for developing a user-targeted product application.

Fundamentally, the ADT consists of the following:

- An architecture-specific cross-toolchain and matching sysroot both built by the OpenEmbedded build system [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#build-system-term]. The toolchain and sysroot are based on a Metadata [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#metadata] configuration and extensions, which allows you to cross-develop on the host machine for the target hardware.
- The Eclipse IDE Yocto Plug-in.
- The Quick EMUlator (QEMU), which lets you simulate target hardware.
- Various user-space tools that greatly enhance your application development experience.

2.1. The Cross-Development Toolchain

The Cross-Development Toolchain [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#cross-development-toolchain] consists of a cross-compiler, cross-linker, and cross-debugger that are used to develop user-space applications for targeted hardware. This toolchain is created either by running the ADT Installer script, a toolchain installer script, or through a Build Directory [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#build-directory] that is based on your Metadata configuration or extension for your targeted device. The cross-toolchain works with a matching target sysroot.

2.2. Sysroot

The matching target sysroot contains needed headers and libraries for generating binaries that run on the target architecture. The sysroot is based on the target root filesystem image that is built by the OpenEmbedded build system and uses the same Metadata configuration used to build the cross-toolchain.

2.3. Eclipse Yocto Plug-in

The Eclipse IDE is a popular development environment and it fully supports development using the Yocto Project. When you install and configure the Eclipse Yocto Project Plug-in into the Eclipse IDE, you maximize your Yocto Project experience. Installing and configuring the Plug-in results in an environment that has extensions specifically designed to let you more easily develop software. These extensions allow for cross-compilation, deployment, and execution of your output into a QEMU emulation session. You can also perform cross-debugging and profiling. The environment also supports a suite of tools that allows you to perform remote profiling, tracing, collection of power data, collection of latency data, and collection of performance data.

For information about the application development workflow that uses the Eclipse IDE and for a detailed example of how to install and configure the Eclipse Yocto Project Plug-in, see the "Working Within Eclipse [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#adt-eclipse]" section of the Yocto Project Development Manual.

2.4. The QEMU Emulator

The QEMU emulator allows you to simulate your hardware while running your application or image. QEMU is made available a number of ways:

- If you use the ADT Installer script to install ADT, you can specify whether or not to install QEMU.

- If you have cloned the poky Git repository to create a Source Directory [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#source-directory] and you have sourced the environment setup script, QEMU is installed and automatically available.
- If you have downloaded a Yocto Project release and unpacked it to create a Source Directory [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#source-directory] and you have sourced the environment setup script, QEMU is installed and automatically available.
- If you have installed the cross-toolchain tarball and you have sourced the toolchain's setup environment script, QEMU is also installed and automatically available.

2.5. User-Space Tools

User-space tools are included as part of the Yocto Project. You will find these tools helpful during development. The tools include LatencyTOP, PowerTOP, OProfile, Perf, SystemTap, and Lttng-ust. These tools are common development tools for the Linux platform.

- LatencyTOP: LatencyTOP focuses on latency that causes skips in audio, stutters in your desktop experience, or situations that overload your server even when you have plenty of CPU power left.
- PowerTOP: Helps you determine what software is using the most power. You can find out more about PowerTOP at https://01.org/powertop/.
- OProfile: A system-wide profiler for Linux systems that is capable of profiling all running code at low overhead. You can find out more about OProfile at http://oprofile.sourceforge.net/about/. For examples on how to setup and use this tool, see the "OProfile [http://www.yoctoproject.org/docs/1.9/profile-manual/profile-manual.html#profile-manual-oprofile]" section in the Yocto Project Profiling and Tracing Manual.
- Perf: Performance counters for Linux used to keep track of certain types of hardware and software events. For more information on these types of counters see https://perf.wiki.kernel.org/. For examples on how to setup and use this tool, see the "perf [http://www.yoctoproject.org/docs/1.9/profile-manual/profile-manual.html#profile-manual-perf]" section in the Yocto Project Profiling and Tracing Manual.
- SystemTap: A free software infrastructure that simplifies information gathering about a running Linux system. This information helps you diagnose performance or functional problems. SystemTap is not available as a user-space tool through the Eclipse IDE Yocto Plug-in. See http://sourceware.org/systemtap for more information on SystemTap. For examples on how to setup and use this tool, see the "SystemTap [http://www.yoctoproject.org/docs/1.9/profile-manual/profile-manual.html#profile-manual-systemtap]" section in the Yocto Project Profiling and Tracing Manual.
- Lttng-ust: A User-space Tracer designed to provide detailed information on user-space activity. See http://lttng.org/ust for more information on Lttng-ust.

Chapter 3. Preparing for Application Development

In order to develop applications, you need set up your host development system. Several ways exist that allow you to install cross-development tools, QEMU, the Eclipse Yocto Plug-in, and other tools. This chapter describes how to prepare for application development.

3.1. Installing the ADT and Toolchains

The following list describes installation methods that set up varying degrees of tool availability on your system. Regardless of the installation method you choose, you must `source` the cross-toolchain environment setup script, which establishes several key environment variables, before you use a toolchain. See the "Setting Up the Cross-Development Environment" section for more information.

Note

Avoid mixing installation methods when installing toolchains for different architectures. For example, avoid using the ADT Installer to install some toolchains and then hand-installing cross-development toolchains by running the toolchain installer for different architectures. Mixing installation methods can result in situations where the ADT Installer becomes unreliable and might not install the toolchain.

If you must mix installation methods, you might avoid problems by deleting `/var/lib/opkg`, thus purging the opkg package metadata.

- Use the ADT installer script: This method is the recommended way to install the ADT because it automates much of the process for you. For example, you can configure the installation to install the QEMU emulator and the user-space NFS, specify which root filesystem profiles to download, and define the target sysroot location.
- Use an existing toolchain: Using this method, you select and download an architecture-specific toolchain installer and then run the script to hand-install the toolchain. If you use this method, you just get the cross-toolchain and QEMU - you do not get any of the other mentioned benefits had you run the ADT Installer script.
- Use the toolchain from within the Build Directory: If you already have a Build Directory [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#build-directory], you can build the cross-toolchain within the directory. However, like the previous method mentioned, you only get the cross-toolchain and QEMU - you do not get any of the other benefits without taking separate steps.

3.1.1. Using the ADT Installer

To run the ADT Installer, you need to get the ADT Installer tarball, be sure you have the necessary host development packages that support the ADT Installer, and then run the ADT Installer Script.

For a list of the host packages needed to support ADT installation and use, see the "ADT Installer Extras" lists in the "Required Packages for the Host Development System [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#required-packages-for-the-host-development-system]" section of the Yocto Project Reference Manual.

3.1.1.1. Getting the ADT Installer Tarball

The ADT Installer is contained in the ADT Installer tarball. You can get the tarball using either of these methods:

- Download the Tarball: You can download the tarball from http://downloads.yoctoproject.org/releases/yocto/yocto-1.9/adt-installer into any directory.
- Build the Tarball: You can use BitBake [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#bitbake-term] to generate the tarball inside an existing Build Directory [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#build-directory].

If you use BitBake to generate the ADT Installer tarball, you must `source` the environment setup script (`oe-init-build-env` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#structure-core-script] or `oe-init-build-env-memres` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#structure-memres-core-script]) located in the Source Directory before running the `bitbake` command that creates the tarball.

The following example commands establish the Source Directory [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#source-directory], check out the current release branch, set up the build environment while also creating the default Build Directory, and run the `bitbake` command that results in the tarball `poky/build/tmp/deploy/sdk/adt_installer.tar.bz2`:

> **Note**
> Before using BitBake to build the ADT tarball, be sure to make sure your `local.conf` file is properly configured. See the "User Configuration [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#user-configuration]" section in the Yocto Project Reference Manual for general configuration information.

```
$ cd ~
$ git clone git://git.yoctoproject.org/poky
$ cd poky
$ git checkout -b tbd origin/tbd
$ source oe-init-build-env
$ bitbake adt-installer
```

3.1.1.2. Configuring and Running the ADT Installer Script

Before running the ADT Installer script, you need to unpack the tarball. You can unpack the tarball in any directory you wish. For example, this command copies the ADT Installer tarball from where it was built into the home directory and then unpacks the tarball into a top-level directory named `adt-installer`:

```
$ cd ~
$ cp poky/build/tmp/deploy/sdk/adt_installer.tar.bz2 $HOME
$ tar -xjf adt_installer.tar.bz2
```

Unpacking it creates the directory `adt-installer`, which contains the ADT Installer script (`adt_installer`) and its configuration file (`adt_installer.conf`).

Before you run the script, however, you should examine the ADT Installer configuration file and be sure you are going to get what you want. Your configurations determine which kernel and filesystem image are downloaded.

The following list describes the configurations you can define for the ADT Installer. For configuration values and restrictions, see the comments in the `adt-installer.conf` file:

- `YOCTOADT_REPO`: This area includes the IPKG-based packages and the root filesystem upon which the installation is based. If you want to set up your own IPKG repository pointed to by `YOCTOADT_REPO`, you need to be sure that the directory structure follows the same layout as the reference directory set up at http://adtrepo.yoctoproject.org. Also, your repository needs to be accessible through HTTP.
- `YOCTOADT_TARGETS`: The machine target architectures for which you want to set up cross-development environments.
- `YOCTOADT_QEMU`: Indicates whether or not to install the emulator QEMU.
- `YOCTOADT_NFS_UTIL`: Indicates whether or not to install user-mode NFS. If you plan to use the Eclipse IDE Yocto plug-in against QEMU, you should install NFS.

> **Note**
> To boot QEMU images using our userspace NFS server, you need to be running `portmap` or `rpcbind`. If you are running `rpcbind`, you will also need to add the `-i` option when `rpcbind`

starts up. Please make sure you understand the security implications of doing this. You might also have to modify your firewall settings to allow NFS booting to work.

- YOCTOADT_ROOTFS_arch: The root filesystem images you want to download from the YOCTOADT_IPKG_REPO repository.
- YOCTOADT_TARGET_SYSROOT_IMAGE_arch: The particular root filesystem used to extract and create the target sysroot. The value of this variable must have been specified with YOCTOADT_ROOTFS_arch. For example, if you downloaded both minimal and sato-sdk images by setting YOCTOADT_ROOTFS_arch to "minimal sato-sdk", then YOCTOADT_ROOTFS_arch must be set to either "minimal" or "sato-sdk".
- YOCTOADT_TARGET_SYSROOT_LOC_arch: The location on the development host where the target sysroot is created.

After you have configured the adt_installer.conf file, run the installer using the following command:

```
$ cd adt-installer
$ ./adt_installer
```

Once the installer begins to run, you are asked to enter the location for cross-toolchain installation. The default location is /opt/poky/release. After either accepting the default location or selecting your own location, you are prompted to run the installation script interactively or in silent mode. If you want to closely monitor the installation, choose “I” for interactive mode rather than “S” for silent mode. Follow the prompts from the script to complete the installation.

Once the installation completes, the ADT, which includes the cross-toolchain, is installed in the selected installation directory. You will notice environment setup files for the cross-toolchain in the installation directory, and image tarballs in the adt-installer directory according to your installer configurations, and the target sysroot located according to the YOCTOADT_TARGET_SYSROOT_LOC_arch variable also in your configuration file.

3.1.2. Using a Cross-Toolchain Tarball

If you want to simply install a cross-toolchain by hand, you can do so by running the toolchain installer. The installer includes the pre-built cross-toolchain, the runqemu script, and support files. If you use this method to install the cross-toolchain, you might still need to install the target sysroot by installing and extracting it separately. For information on how to install the sysroot, see the "Extracting the Root Filesystem" section.

Follow these steps:

1. Get your toolchain installer using one of the following methods:

   - Go to http://downloads.yoctoproject.org/releases/yocto/yocto-1.9/toolchain/ and find the folder that matches your host development system (i.e. i686 for 32-bit machines or x86_64 for 64-bit machines).

     Go into that folder and download the toolchain installer whose name includes the appropriate target architecture. The toolchains provided by the Yocto Project are based off of the core-image-sato image and contain libraries appropriate for developing against that image. For example, if your host development system is a 64-bit x86 system and you are going to use your cross-toolchain for a 32-bit x86 target, go into the x86_64 folder and download the following installer:

     ```
     poky-glibc-x86_64-core-image-sato-i586-toolchain-1.9.sh
     ```

   - Build your own toolchain installer. For cases where you cannot use an installer from the download area, you can build your own as described in the "Optionally Building a Toolchain Installer" section.

2. Once you have the installer, run it to install the toolchain:

Note

You must change the permissions on the toolchain installer script so that it is executable.

The following command shows how to run the installer given a toolchain tarball for a 64-bit x86 development host system and a 32-bit x86 target architecture. The example assumes the toolchain installer is located in `~/Downloads/`.

```
$ ~/Downloads/poky-glibc-x86_64-core-image-sato-i586-toolchain-1.9.sh
```

The first thing the installer prompts you for is the directory into which you want to install the toolchain. The default directory used is `/opt/poky/1.9`. If you do not have write permissions for the directory into which you are installing the toolchain, the toolchain installer notifies you and exits. Be sure you have write permissions in the directory and run the installer again.

When the script finishes, the cross-toolchain is installed. You will notice environment setup files for the cross-toolchain in the installation directory.

3.1.3. Using BitBake and the Build Directory

A final way of making the cross-toolchain available is to use BitBake to generate the toolchain within an existing Build Directory [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#build-directory]. This method does not install the toolchain into the default `/opt` directory. As with the previous method, if you need to install the target sysroot, you must do that separately as well.

Follow these steps to generate the toolchain into the Build Directory:

1. Set up the Build Environment: Source the OpenEmbedded build environment setup script (i.e. `oe-init-build-env` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#structure-core-script] or `oe-init-build-env-memres` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#structure-memres-core-script]) located in the Source Directory [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#source-directory].

2. Check your Local Configuration File: At this point, you should be sure that the `MACHINE` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-MACHINE] variable in the `local.conf` file found in the `conf` directory of the Build Directory is set for the target architecture. Comments within the `local.conf` file list the values you can use for the `MACHINE` variable.

   ### Note

   You can populate the Build Directory with the cross-toolchains for more than a single architecture. You just need to edit the `MACHINE` variable in the `local.conf` file and re-run the `bitbake` command.

3. Generate the Cross-Toolchain: Run `bitbake meta-ide-support` to complete the cross-toolchain generation. Once the `bitbake` command finishes, the cross-toolchain is generated and populated within the Build Directory. You will notice environment setup files for the cross-toolchain that contain the string "`environment-setup`" in the Build Directory's `tmp` folder.

   Be aware that when you use this method to install the toolchain, you still need to separately extract and install the sysroot filesystem. For information on how to do this, see the "Extracting the Root Filesystem" section.

3.2. Setting Up the Cross-Development Environment

Before you can develop using the cross-toolchain, you need to set up the cross-development environment by sourcing the toolchain's environment setup script. If you used the ADT Installer or hand-installed cross-toolchain, then you can find this script in the directory you chose for installation. For this release, the default installation directory is `/opt/poky/1.9`. If you installed the toolchain in the Build Directory [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#build-

directory], you can find the environment setup script for the toolchain in the Build Directory's `tmp` directory.

Be sure to run the environment setup script that matches the architecture for which you are developing. Environment setup scripts begin with the string "`environment-setup`" and include as part of their name the architecture. For example, the toolchain environment setup script for a 64-bit IA-based architecture installed in the default installation directory would be the following:

```
/opt/poky/1.9/environment-setup-x86_64-poky-linux
```

When you run the setup script, many environment variables are defined:

```
SDKTARGETSYSROOT [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-SDKTAR
PKG_CONFIG_PATH [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-PKG_CON
CONFIG_SITE [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-CONFIG_SITE
CC [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-CC] - The minimal co
CXX [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-CXX] - The minimal
CPP [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-CPP] - The minimal
AS [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-AS] - The minimal co
LD [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-LD] - The minimal co
GDB [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-GDB] - The minimal
STRIP [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-STRIP] - The mini
RANLIB [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-RANLIB] - The mi
OBJCOPY [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-OBJCOPY] - The
OBJDUMP [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-OBJDUMP] - The
AR [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-AR] - The minimal co
NM [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-NM] - The minimal co
TARGET_PREFIX [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-TARGET_PR
CROSS_COMPILE [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-CROSS_COM
CONFIGURE_FLAGS [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-CONFIGU
CFLAGS [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-CFLAGS] - Sugges
CXXFLAGS [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-CXXFLAGS] - Su
LDFLAGS [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-LDFLAGS] - Sugg
CPPFLAGS [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-CPPFLAGS] - Su
```

3.3. Securing Kernel and Filesystem Images

You will need to have a kernel and filesystem image to boot using your hardware or the QEMU emulator. Furthermore, if you plan on booting your image using NFS or you want to use the root filesystem as the target sysroot, you need to extract the root filesystem.

3.3.1. Getting the Images

To get the kernel and filesystem images, you either have to build them or download pre-built versions. You can find examples for both these situations in the "A Quick Test Run [http://www.yoctoproject.org/docs/1.9/yocto-project-qs/yocto-project-qs.html#test-run]" section of the Yocto Project Quick Start.

The Yocto Project ships basic kernel and filesystem images for several architectures (x86, `x86-64`, `mips`, `powerpc`, and `arm`) that you can use unaltered in the QEMU emulator. These kernel images reside in the release area - http://downloads.yoctoproject.org/releases/yocto/yocto-1.9/machines and are ideal for experimentation using Yocto Project. For information on the image types you can build using the OpenEmbedded build system, see the "Images [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#ref-images]" chapter in the Yocto Project Reference Manual.

If you are planning on developing against your image and you are not building or using one of the Yocto Project development images (e.g. `core-image-*-dev`), you must be sure to include the development packages as part of your image recipe.

Furthermore, if you plan on remotely deploying and debugging your application from within the Eclipse IDE, you must have an image that contains the Yocto Target Communication Framework

(TCF) agent (`tcf-agent`). By default, the Yocto Project provides only one type of pre-built image that contains the `tcf-agent`. And, those images are SDK (e.g.`core-image-sato-sdk`).

If you want to use a different image type that contains the `tcf-agent`, you can do so one of two ways:

- Modify the `conf/local.conf` configuration in the Build Directory [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#build-directory] and then rebuild the image. With this method, you need to modify the EXTRA_IMAGE_FEATURES [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-EXTRA_IMAGE_FEATURES] variable to have the value of "tools-debug" before rebuilding the image. Once the image is rebuilt, the `tcf-agent` will be included in the image and is launched automatically after the boot.
- Manually build the `tcf-agent`. To build the agent, follow these steps:
  1. Be sure the ADT is installed as described in the "Installing the ADT and Toolchains" section.
  2. Set up the cross-development environment as described in the "Setting Up the Cross-Development Environment" section.
  3. Get the `tcf-agent` source code using the following commands:

     ```
     $ git clone http://git.eclipse.org/gitroot/tcf/org.eclipse.tcf.agent.git
     $ cd org.eclipse.tcf.agent/agent
     ```

  4. Locate the `Makefile.inc` file inside the `agent` folder and modify it for the cross-compilation environment by setting the `OPSYS` and `MACHINE` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-MACHINE] variables according to your target.
  5. Use the cross-development tools to build the `tcf-agent`. Before you "Make" the file, be sure your cross-tools are set up first. See the "Makefile-Based Projects" section for information on how to make sure the cross-tools are set up correctly.

     If the build is successful, the `tcf-agent` output will be `obj/$(OPSYS)/$(MACHINE)/Debug/agent`.
  6. Deploy the agent into the image's root filesystem.

3.3.2. Extracting the Root Filesystem

If you install your toolchain by hand or build it using BitBake and you need a root filesystem, you need to extract it separately. If you use the ADT Installer to install the ADT, the root filesystem is automatically extracted and installed.

Here are some cases where you need to extract the root filesystem:

- You want to boot the image using NFS.
- You want to use the root filesystem as the target sysroot. For example, the Eclipse IDE environment with the Eclipse Yocto Plug-in installed allows you to use QEMU to boot under NFS.
- You want to develop your target application using the root filesystem as the target sysroot.

To extract the root filesystem, first `source` the cross-development environment setup script to establish necessary environment variables. If you built the toolchain in the Build Directory, you will find the toolchain environment script in the `tmp` directory. If you installed the toolchain by hand, the environment setup script is located in `/opt/poky/1.9`.

After sourcing the environment script, use the `runqemu-extract-sdk` command and provide the filesystem image.

Following is an example. The second command sets up the environment. In this case, the setup script is located in the `/opt/poky/1.9` directory. The third command extracts the root filesystem from a previously built filesystem that is located in the `~/Downloads` directory. Furthermore, this command extracts the root filesystem into the `qemux86-sato` directory:

```
$ cd ~
```

```
$ source /opt/poky/1.9/environment-setup-i586-poky-linux
$ runqemu-extract-sdk \
   ~/Downloads/core-image-sato-sdk-qemux86-2011091411831.rootfs.tar.bz2 \
   $HOME/qemux86-sato
```

You could now point to the target sysroot at `qemux86-sato`.

3.4. Optionally Building a Toolchain Installer

As an alternative to locating and downloading a toolchain installer, you can build the toolchain installer one of two ways if you have a Build Directory [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#build-directory]:

- Use `bitbake meta-toolchain`. This method requires you to still install the target sysroot by installing and extracting it separately. For information on how to install the sysroot, see the "Extracting the Root Filesystem" section.
- Use `bitbake image -c populate_sdk`. This method has significant advantages over the previous method because it results in a toolchain installer that contains the sysroot that matches your target root filesystem.

  Another powerful feature is that the toolchain is completely self-contained. The binaries are linked against their own copy of `libc`, which results in no dependencies on the target system. To achieve this, the pointer to the dynamic loader is configured at install time since that path cannot be dynamically altered. This is the reason for a wrapper around the `populate_sdk` archive.

  Another feature is that only one set of cross-canadian toolchain binaries are produced per architecture. This feature takes advantage of the fact that the target hardware can be passed to `gcc` as a set of compiler options. Those options are set up by the environment script and contained in variables such as `CC` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-CC] and `LD` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-LD]. This reduces the space needed for the tools. Understand, however, that a sysroot is still needed for every target since those binaries are target-specific.

Remember, before using any BitBake command, you must source the build environment setup script (i.e. `oe-init-build-env` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#structure-core-script] or `oe-init-build-env-memres` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#structure-memres-core-script]) located in the Source Directory and you must make sure your `conf/local.conf` variables are correct. In particular, you need to be sure the `MACHINE` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-MACHINE] variable matches the architecture for which you are building and that the `SDKMACHINE` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-SDKMACHINE] variable is correctly set if you are building a toolchain designed to run on an architecture that differs from your current development host machine (i.e. the build machine).

When the `bitbake` command completes, the toolchain installer will be in `tmp/deploy/sdk` in the Build Directory.

Note

By default, this toolchain does not build static binaries. If you want to use the toolchain to build these types of libraries, you need to be sure your image has the appropriate static development libraries. Use the `IMAGE_INSTALL` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-IMAGE_INSTALL] variable inside your `local.conf` file to install the appropriate library packages. Following is an example using `glibc` static development libraries:

```
IMAGE_INSTALL_append = " glibc-staticdev"
```

3.5. Optionally Using an External Toolchain

You might want to use an external toolchain as part of your development. If this is the case, the fundamental steps you need to accomplish are as follows:

- Understand where the installed toolchain resides. For cases where you need to build the external toolchain, you would need to take separate steps to build and install the toolchain.
- Make sure you add the layer that contains the toolchain to your `bblayers.conf` file through the `BBLAYERS` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-BBLAYERS] variable.
- Set the `EXTERNAL_TOOLCHAIN` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-EXTERNAL_TOOLCHAIN] variable in your `local.conf` file to the location in which you installed the toolchain.

A good example of an external toolchain used with the Yocto Project is Mentor Graphics® Sourcery G ++ Toolchain. You can see information on how to use that particular layer in the `README` file at http://github.com/MentorEmbedded/meta-sourcery/. You can find further information by reading about the `TCMODE` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-TCMODE] variable in the Yocto Project Reference Manual's variable glossary.

Chapter 4. Optionally Customizing the Development Packages Installation

Because the Yocto Project is suited for embedded Linux development, it is likely that you will need to customize your development packages installation. For example, if you are developing a minimal image, then you might not need certain packages (e.g. graphics support packages). Thus, you would like to be able to remove those packages from your target sysroot.

4.1. Package Management Systems

The OpenEmbedded build system supports the generation of sysroot files using three different Package Management Systems (PMS):

- OPKG: A less well known PMS whose use originated in the OpenEmbedded and OpenWrt embedded Linux projects. This PMS works with files packaged in an `.ipk` format. See http://en.wikipedia.org/wiki/Opkg for more information about OPKG.
- RPM: A more widely known PMS intended for GNU/Linux distributions. This PMS works with files packaged in an `.rms` format. The build system currently installs through this PMS by default. See http://en.wikipedia.org/wiki/RPM_Package_Manager for more information about RPM.
- Debian: The PMS for Debian-based systems is built on many PMS tools. The lower-level PMS tool dpkg forms the base of the Debian PMS. For information on dpkg see http://en.wikipedia.org/wiki/Dpkg.

4.2. Configuring the PMS

Whichever PMS you are using, you need to be sure that the PACKAGE_CLASSES [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-PACKAGE_CLASSES] variable in the `conf/local.conf` file is set to reflect that system. The first value you choose for the variable specifies the package file format for the root filesystem at sysroot. Additional values specify additional formats for convenience or testing. See the `conf/local.conf` configuration file for details.

> **Note**
> For build performance information related to the PMS, see the "`package.bbclass` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#ref-classes-package]" section in the Yocto Project Reference Manual.

As an example, consider a scenario where you are using OPKG and you want to add the `libglade` package to the target sysroot.

First, you should generate the IPK file for the `libglade` package and add it into a working `opkg` repository. Use these commands:

```
$ bitbake libglade
$ bitbake package-index
```

Next, source the cross-toolchain environment setup script found in the Source Directory [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#source-directory]. Follow that by setting up the installation destination to point to your sysroot as `sysroot_dir`. Finally, have an OPKG configuration file `conf_file` that corresponds to the `opkg` repository you have just created. The following command forms should now work:

```
$ opkg-cl –f conf_file -o sysroot_dir update
$ opkg-cl –f cconf_file -o sysroot_dir \
```

```
   --force-overwrite install libglade
$ opkg-cl –f cconf_file -o sysroot_dir \
   --force-overwrite install libglade-dbg
$ opkg-cl –f conf_file> -o sysroot_dir> \
   --force-overwrite install libglade-dev
```

Chapter 5. Using the Command Line

Recall that earlier the manual discussed how to use an existing toolchain tarball that had been installed into the default installation directory, `/opt/poky/1.9`, which is outside of the Build Directory [http://www.yoctoproject.org/docs/1.9/dev-manual/dev-manual.html#build-directory] (see the section "Using a Cross-Toolchain Tarball)". And, that sourcing your architecture-specific environment setup script initializes a suitable cross-toolchain development environment.

During this setup, locations for the compiler, QEMU scripts, QEMU binary, a special version of `pkgconfig` and other useful utilities are added to the PATH variable. Also, variables to assist `pkgconfig` and `autotools` are also defined so that, for example, `configure.sh` can find pre-generated test results for tests that need target hardware on which to run. You can see the "Setting Up the Cross-Development Environment" section for the list of cross-toolchain environment variables established by the script.

Collectively, these conditions allow you to easily use the toolchain outside of the OpenEmbedded build environment on both Autotools-based projects and Makefile-based projects. This chapter provides information for both these types of projects.

5.1. Autotools-Based Projects

Once you have a suitable cross-toolchain installed, it is very easy to develop a project outside of the OpenEmbedded build system. This section presents a simple "Helloworld" example that shows how to set up, compile, and run the project.

5.1.1. Creating and Running a Project Based on GNU Autotools

Follow these steps to create a simple Autotools-based project:

1. Create your directory: Create a clean directory for your project and then make that directory your working location:

```
$ mkdir $HOME/helloworld
$ cd $HOME/helloworld
```

2. Populate the directory: Create `hello.c`, `Makefile.am`, and `configure.in` files as follows:

   - For `hello.c`, include these lines:

```
#include <stdio.h>

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

   - For `Makefile.am`, include these lines:

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

   - For `configure.in`, include these lines:

```
AC_INIT(hello.c)
AM_INIT_AUTOMAKE(hello,0.1)
AC_PROG_CC
AC_PROG_INSTALL
AC_OUTPUT(Makefile)
```

3. Source the cross-toolchain environment setup file: Installation of the cross-toolchain creates a cross-toolchain environment setup script in the directory that the ADT was installed. Before you can use the tools to develop your project, you must source this setup script. The script begins with the string "environment-setup" and contains the machine architecture, which is followed by the string "poky-linux". Here is an example that sources a script from the default ADT installation directory that uses the 32-bit Intel x86 Architecture and the tbd Yocto Project release:

```
$ source /opt/poky/1.9/environment-setup-i586-poky-linux
```

4. Generate the local aclocal.m4 files and create the configure script: The following GNU Autotools generate the local `aclocal.m4` files and create the configure script:

```
$ aclocal
$ autoconf
```

5. Generate files needed by GNU coding standards: GNU coding standards require certain files in order for the project to be compliant. This command creates those files:

```
$ touch NEWS README AUTHORS ChangeLog
```

6. Generate the configure file: This command generates the `configure`:

```
$ automake -a
```

7. Cross-compile the project: This command compiles the project using the cross-compiler. The `CONFIGURE_FLAGS` [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-CONFIGURE_FLAGS] environment variable provides the minimal arguments for GNU configure:

```
$ ./configure ${CONFIGURE_FLAGS}
```

8. Make and install the project: These two commands generate and install the project into the destination directory:

```
$ make
$ make install DESTDIR=./tmp
```

9. Verify the installation: This command is a simple way to verify the installation of your project. Running the command prints the architecture on which the binary file can run. This architecture should be the same architecture that the installed cross-toolchain supports.

```
$ file ./tmp/usr/local/bin/hello
```

10. Execute your project: To execute the project in the shell, simply enter the name. You could also copy the binary to the actual target hardware and run the project there as well:

```
$ ./hello
```

As expected, the project displays the "Hello World!" message.

5.1.2. Passing Host Options

For an Autotools-based project, you can use the cross-toolchain by just passing the appropriate host option to `configure.sh`. The host option you use is derived from the name of the environment setup script found in the directory in which you installed the cross-toolchain. For example, the host option for an ARM-based target that uses the GNU EABI is `armv5te-poky-linux-gnueabi`. You will notice that the name of the script is `environment-setup-armv5te-poky-linux-gnueabi`. Thus, the following command works to update your project and rebuild it using the appropriate cross-toolchain tools:

```
$ ./configure --host=armv5te-poky-linux-gnueabi \
   --with-libtool-sysroot=sysroot_dir
```

Note

If the `configure` script results in problems recognizing the `--with-libtool-sysroot=sysroot-dir` option, regenerate the script to enable the support by doing the following and then run the script again:

```
$ libtoolize --automake
$ aclocal -I ${OECORE_NATIVE_SYSROOT}/usr/share/aclocal \
   [-I dir_containing_your_project-specific_m4_macros]
$ autoconf
$ autoheader
$ automake -a
```

5.2. Makefile-Based Projects

For Makefile-based projects, the cross-toolchain environment variables established by running the cross-toolchain environment setup script are subject to general `make` rules.

To illustrate this, consider the following four cross-toolchain environment variables:

```
CC [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-CC]=i586-poky-linux-
LD [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-LD]=i586-poky-linux-
CFLAGS [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-CFLAGS]=-O2 -pip
CXXFLAGS [http://www.yoctoproject.org/docs/1.9/ref-manual/ref-manual.html#var-CXXFLAGS]=-O2
```

Now, consider the following three cases:

- Case 1 - No Variables Set in the `Makefile`: Because these variables are not specifically set in the `Makefile`, the variables retain their values based on the environment.
- Case 2 - Variables Set in the `Makefile`: Specifically setting variables in the `Makefile` during the build results in the environment settings of the variables being overwritten.
- Case 3 - Variables Set when the `Makefile` is Executed from the Command Line: Executing the `Makefile` from the command line results in the variables being overwritten with command-line content regardless of what is being set in the `Makefile`. In this case, environment variables are not considered unless you use the "-e" flag during the build:

  ```
  $ make -e file
  ```

  If you use this flag, then the environment values of the variables override any variables specifically set in the `Makefile`.

Note

For the list of variables set up by the cross-toolchain environment setup script, see the "Setting Up the Cross-Development Environment" section.