Application Development Toolkit (ADT) User's Guide

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

by Jessica Zhang

Table of Contents

1. Application Development Toolkit (ADT) User's Guide 1
    1.1. Introducing the Application Development Toolkit (ADT) 1
        1.1.1. The Cross-Toolchain 1
        1.1.2. Sysroot 1
        1.1.3. The QEMU Emulator 1
        1.1.4. User-Space Tools 1
2. Preparing to Use the Application Development Toolkit (ADT) 3
    2.1. Installing the ADT 3
        2.1.1. Configuring and Running the ADT Installer 3
        2.1.2. Using an Existing Toolchain Tarball 4
        2.1.3. Using the Toolchain from Within the Build Tree 4
    2.2. Setting Up the Environment 5
    2.3. Kernels and Filesystem Images 5
3. Optionally Customizing the Development Packages Installation 6
    3.1. Package Management Systems 6
    3.2. Configuring the PMS 6
4. Working Within Eclipse 7
    4.1. Setting Up the Eclipse IDE 7
        4.1.1. Installing Eclipse IDE 7
        4.1.2. Installing Required Plug-ins and the Eclipse Yocto Plug-in 7
        4.1.3. Configuring the Plug-in 8
    4.2. Creating the Project 9
    4.3. Configuring the Cross-Toolchains 9
    4.4. Building the Project 10
    4.5. Starting QEMU in User Space NFS Mode 10
    4.6. Deploying and Debugging the Application 10
    4.7. Running User-Space Tools 11
5. Using the Command Line 12
    5.1. Autotools-Based Projects 12
    5.2. Makefile-Based Projects 12

Chapter 1. Application Development Toolkit (ADT) User's Guide

Welcome to the Application Development Toolkit User’s Guide. This manual provides information that lets you get going with the ADT to develop projects using the Yocto Project.

1.1. Introducing the Application Development Toolkit (ADT)

Fundamentally, the ADT consists of an architecture-specific cross-toolchain and a matching sysroot that are both built by the Poky build system. The toolchain and sysroot are based on a metadata configuration and extensions, which allows you to cross develop for the target on the host machine.

Additionally, to provide an effective development platform, the Yocto Project makes available and suggests other tools as part of the ADT. These other tools include the Eclipse IDE Yocto Plug-in, an emulator (QEMU), and various user-space tools that greatly enhance your development experience.

The resulting combination of the architecture-specific cross-toolchain and sysroot along with these additional tools yields a custom-built, cross-development platform for a user-targeted product.

1.1.1. The Cross-Toolchain

The cross-toolchain consists of a cross-compiler, cross-linker, and cross-debugger that are all generated through a Poky build that is based on your metadata configuration or extension for your targeted device. The cross-toolchain works with a matching target sysroot.

1.1.2. 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 Poky and uses the same metadata configuration used to build the cross-toolchain.

1.1.3. The QEMU Emulator

The QEMU emulator allows you to simulate your hardware while running your application or image. QEMU is installed several ways: as part of the Poky tree, ADT installation through a toolchain tarball, or through the ADT Installer.

1.1.4. User-Space Tools

User-space tools are included as part of the distribution. 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. You can find out more about LatencyTOP at http://www.latencytop.org/.
- PowerTOP – Helps you determine what software is using the most power. You can find out more about PowerTOP at http://www.linuxpowertop.org/.
- 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/.
- 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/index.php and click on “Perf tools.”

- 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 Yocto Eclipse IDE Plug-in. See http://sourceware.org/systemtap for more information on SystemTap.
- 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 2. Preparing to Use the Application Development Toolkit (ADT)

In order to use the ADT it must be installed, the environment setup script must be sourced, and the kernel and filesystem image specific to the target architecture must exist. This section describes how to install the ADT, set up the environment, and provides some reference information on kernels and filesystem images.

2.1. Installing the ADT

You can install the ADT three ways. However, we recommend configuring and running the ADT Installer script. Running this script automates much of the process for you. For example, the script allows you to install the QEMU emulator and user-space NFS, define which root filesystem profiles to download, and allows you to define the target sysroot location.

Note

If you need to generate the ADT tarball you can do so using the following command:

```
$ bitbake adt-installer
```

This command generates the file `adt-installer.tar.bz2` in the `../build/tmp/deploy/sdk` directory.

2.1.1. Configuring and Running the ADT Installer

The ADT Installer is contained in a tarball that can be built using `bitbake adt-installer`. Yocto Project has a pre-built ADT Installer tarball that you can download from `tmp/deploy/sdk` located in the build directory.

Note

You can install and run the ADT Installer tarball in any directory you want.

Before running the ADT Installer you need to configure it by editing the `adt-installer.conf` file, which is located in the directory where the ADT Installer tarball was installed. Your configurations determine which kernel and filesystem image are downloaded. The following list describes the variables you can define for the ADT Installer. For configuration values and restrictions see the comments in the `adt-installer.conf` file:

- `YOCTOADT_IPKG_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_IPKG_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 Yocto Eclipse IDE 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. Your firewall settings may also have to be modified to allow NFS booting to work.

- YOCTOADT_ROOTFS_<arch> - The root filesystem images you want to download.
- YOCTOADT_TARGET_SYSROOT_IMAGE_<arch> - The root filesystem used to extract and create the target sysroot.
- YOCTOADT_TARGET_SYSROOT_LOC_<arch> - The location of the target sysroot that will be set up on the development machine.

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

```
$ adt_installer
```

Once the installer begins to run you are asked whether you want to run in interactive or silent mode. If you want to closely monitor the installation then 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 cross-toolchain is installed in /opt/poky/$SDKVERSION.

Before using the ADT you need to run the environment setup script for your target architecture also located in /opt/poky/$SDKVERSION. See the Section 2.2, “Setting Up the Environment”section for information.

2.1.2. Using an Existing Toolchain Tarball

If you do not want to use the ADT Installer you can install the toolchain and the sysroot by hand. Follow these steps:

1. Locate and download the architecture-specific toolchain tarball from http://autobuilder.yoctoproject.org/downloads/yocto-1.0. Look in the ‘toolchain’ folder and then open up the folder that matches your host development system (i.e. 'i686' for 32-bit machines or 'x86_64' for 64-bit machines). Then, select the toolchain tarball whose name includes the appropriate target architecture.

   ### Note

   If you need to build the toolchain tarball use the bitbake meta-toolchain command after you have sourced the poky-build-init script. The tarball will be located in the build directory at tmp/deploy/sdk after the build.

2. Make sure you are in the root directory and then expand the tarball. The tarball expands into the /opt/poky/$SDKVERSION directory.

3. Set up the environment by sourcing the environment set up script. See the Section 2.2, “Setting Up the Environment” for information.

2.1.3. Using the Toolchain from Within the Build Tree

A final way of accessing the toolchain is from the build tree. The build tree can be set up to contain the architecture-specific cross toolchain. To populate the build tree with the toolchain you need to run the following command:

```
$ bitbake meta-ide-support
```

Before running the command you need to be sure that the conf/local.conf file in the build directory has the desired architecture specified for the MACHINE variable. See the local.conf file for a list of values you can supply for this variable. You can populate the build tree with the cross-toolchains for more than a single architecture. You just need to edit the local.conf file and re-run the BitBake command.

Once the build tree has the toolchain you need to source the environment setup script so that you can run the cross-tools without having to locate them. See the Section 2.2, “Setting Up the Environment” for information.

2.2. Setting Up the Environment

Before you can use the cross-toolchain you need to set up the environment by sourcing the environment setup script. If you used adt_installer or used an existing ADT tarball to install the ADT, then you can find this script in the `/opt/poky/$SDKVERSION` directory. If you are using the ADT from a Poky build tree, then look in the build directory in `tmp` for the setup script.

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 environment setup script for a 64-bit IA-based architecture would be the following:

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

2.3. Kernels and Filesystem Images

You will need to have a kernel and filesystem image to boot using your hardware or the QEMU emulator. That means you either have to build them or know where to get them. You can find lots of details on how to get or build images and kernels for your architecture in the "Yocto Project Quick Start" found at http://www.yoctoproject.org/docs/yocto-quick-start/yocto-project-qs.html.

Note

Yocto Project provides basic kernels and filesystem images for several architectures (x86, x86-64, mips, powerpc, and arm) that can be used unaltered in the QEMU emulator. These kernels and filesystem images reside in the Yocto Project release area - http://autobuilder.yoctoproject.org/downloads/yocto-1.0/ and are ideal for experimentation within Yocto Project.

Chapter 3. 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 sysroot.

3.1. Package Management Systems

The Yocto Project supports the generation of root filesystem 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 Yocto Project 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.

3.2. Configuring the PMS

Whichever PMS you are using you need to be sure that the `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. Additional values specify additional formats for convenience or testing. See the configuration file for details.

As an example, consider a scenario where you are using OPKG and you want to add the libglade package to 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 environment setup script. 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 <conf file>> -o <sysroot dir> --force-overwrite install libglade
$ opkg-cl –f <conf file> -o <sysroot dir> --force-overwrite install libglade-dbg
$ opkg-cl –f <conf file> -o <sysroot dir> --force-overwrite install libglade-dev
```

Chapter 4. Working Within Eclipse

The Eclipse IDE is a popular development environment and it fully supports development using Yocto Project. When you install and configure the Eclipse Yocto Project Plug-in into the Eclipse IDE you maximize your Yocto Project design 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 and deployment and execution of your output into a QEMU emulation session. You can also perform cross-debugging and profiling. The environment also has 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.

This section describes how to install and configure the Eclipse IDE Yocto Plug-in and how to use it to develop your Yocto Project.

4.1. Setting Up the Eclipse IDE

To develop within the Eclipse IDE you need to do the following:

1. Be sure the optimal version of Eclipse IDE is installed.

2. Install required Eclipse plug-ins prior to installing the Eclipse Yocto Plug-in.

3. Configure the Eclipse Yocto Plug-in.

4.1.1. Installing Eclipse IDE

It is recommended that you have the Helios 3.6.1 version of the Eclipse IDE installed on your development system. If you don't have this version you can find it at http://www.eclipse.org/downloads. From that site, choose the Eclipse Classic version. This version contains the Eclipse Platform, the Java Development Tools (JDT), and the Plug-in Development Environment.

Once you have downloaded the tarball, extract it into a clean directory and complete the installation.

One issue exists that you need to be aware of regarding the Java Virtual machine's garbage collection (GC) process. The GC process does not clean up the permanent generation space (PermGen). This space stores meta-data descriptions of classes. The default value is set too small and it could trigger an out-of-memory error such as the following:

```
Java.lang.OutOfMemoryError: PermGen space
```

This error causes the application to hang.

To fix this issue you can use the □□vmargs option when you start Eclipse to increase the size of the permanent generation space:

```
eclipse □□vmargs □□XX:PermSize=256M
```

4.1.2. Installing Required Plug-ins and the Eclipse Yocto Plug-in

Before installing the Yocto Plug-in you need to be sure that the CDT 7.0, RSE 3.2, and Autotools plug-ins are all installed in the following order. After installing these three plug-ins, you can install the Eclipse Yocto Plug-in. Use the following URLs for the plug-ins:

1. CDT 7.0 – http://download.eclipse.org/tools/cdt/releases/helios/: For CDT main features select the checkbox so you get all items. For CDT optional features expand the selections and check "C/C++ Remote Launch".

2. RSE 3.2 – http://download.eclipse.org/tm/updates/3.2: Check the box next to "TM and RSE Main Features" so you select all those items. Note that all items in the main features depend

on 3.2.1 version. Expand the items under “TM and RSE Uncategorized 3.2.1” and select the following: “Remote System Explorer End-User Runtime”, “Remote System Explorer Extended SDK”, “Remote System Explorer User Actions”, “RSE Core”, “RSE Terminals UI”, and “Target Management Terminal”.

3. Autotools – http://download.eclipse.org/technology/linuxtools/update/: Expand the items under “Linux Tools” and select “Autotools support for CDT (Incubation)”.

4. Yocto Plug-in – http://www.yoctoproject.org/downloads/eclipse-plugin/1.0: Check the box next to “Development tools & SDKs for Yocto Linux” to select all the items.

Follow these general steps to install a plug-in:

1. From within the Eclipse IDE select the “Install New Software” item from the “Help” menu.

2. Click “Add...” in the “Work with:” area.

3. Enter the URL for the repository and leave the “Name” field blank.

4. Check the boxes next to the software you need to install and then complete the installation. For information on the specific software packages you need to include, see the previous list.

4.1.3. Configuring the Plug-in

Configuring the Eclipse Yocto Plug-in involves choosing the Cross Compiler Options, selecting the Target Architecture, and choosing the Target Options. These settings are the default settings for all projects. You do have opportunities to change them later if you choose to when you configure the project. See “Configuring the Cross Toolchain” section later in the manual.

To start, you need to do the following from within the Eclipse IDE:

- Choose Windows -> Preferences to display the Preferences Dialog
- Click “Yocto SDK”

4.1.3.1. Configuring the Cross-Compiler Options

Choose between ‘SDK Root Mode’ and ‘Poky Tree Mode’ for Cross Compiler Options.

- SDK Root Mode – Select this mode when you are not concerned with building an image or you do not have a Poky build tree on your system. For example, suppose you are an application developer and do not need to build an image. You just want to use an architecture-specific toolchain on an existing kernel and root filesystem. When you use SDK Root Mode you are using the toolchain installed in the `/opt/poky` directory.
- Poky Tree Mode – Select this mode if you are concerned with building images for hardware or your development environment already has a build tree. In this case you likely already have a Poky build tree installed on your system or you (or someone else) will be building one. When you use the Poky Tree Mode you are using the toolchain bundled inside the Poky build tree. If you use this mode you must also supply the Poky Root Location in the Preferences Dialog.

4.1.3.2. Configuring the Sysroot

Specify the sysroot, which is used by both the QEMU user-space NFS boot process and by the cross-toolchain regardless of the mode you select (SDK Root Mode or Poky Tree Mode). For example, sysroot is the location to which you extract the downloaded image’s root filesystem to through the ADT Installer.

4.1.3.3. Selecting the Target Architecture

Use the pull-down Target Architecture menu and select the target architecture.

The Target Architecture is the type of hardware you are going to use or emulate. This pull-down menu should have the supported architectures. If the architecture you need is not listed in the menu then you will need to re-visit Chapter 2,Preparing to Use the Application Development Toolkit (ADT) section earlier in this document.

4.1.3.4. Choosing the Target Options

You can choose to emulate hardware using the QEMU emulator, or you can choose to use actual hardware.

- External HW – Select this option if you will be using actual hardware.
- QEMU – Select this option if you will be using the QEMU emulator. If you are using the emulator you also need to locate the Kernel and you can specify custom options.

  In Poky Tree Mode the kernel you built will be located in the Poky Build tree in `tmp/deploy/images` directory. In SDK Root Mode the pre-built kernel you downloaded is located in the directory you specified when you downloaded the image.

  Most custom options are for advanced QEMU users to further customize their QEMU instance. These options are specified between paired angled brackets. Some options must be specified outside the brackets. In particular, the options `serial`, `nographic`, and `kvm` must all be outside the brackets. Use the `man qemu` command to get help on all the options and their use. The following is an example:

  ```
  serial ‘<-m 256 -full-screen>’
  ```

  Regardless of the mode, Sysroot is already defined in the “Sysroot” field.

Click the “OK” button to save your plug-in configurations.

4.2. Creating the Project

You can create two types of projects: Autotools-based, or Makefile-based. This section describes how to create autotools-based projects from within the Eclipse IDE. For information on creating projects in a terminal window see Chapter 5,Using the Command Line section.

To create a project based on a Yocto template and then display the source code, follow these steps:

1. Select File -> New -> Project.
2. Double click “CC++”.
3. Double click “C Project” to create the project.
4. Double click “Yocto SDK Project”.
5. Select “Hello World ANSI C Autotools Project”. This is an Autotools-based project based on a Yocto Project template.
6. Put a name in the “Project name:” field.
7. Click “Next”.
8. Add information in the “Author” field.
9. Use “GNU General Public License v2.0” for the License.
10. Click “Finish”.
11. Answer ‘Yes” to the open perspective prompt.
12. In the Project Explorer expand your project.
13. Expand ‘src’.
14. Double click on your source file and the code appears in the window. This is the template.

4.3. Configuring the Cross-Toolchains

The previous section, Section 4.1.3.1, “Configuring the Cross-Compiler Options”, set up the default project configurations. You can change these settings for a given project by following these steps:

1. Select Project -> Invoke Yocto Tools -> Reconfigure Yocto. This brings up the project Yocto Settings Dialog. Settings are inherited from the default project configuration. The information in this dialogue is identical to that chosen earlier for the Cross Compiler Option (SDK Root Mode or Poky Tree Mode), the Target Architecture, and the Target Options. The settings are inherited from the Yocto Plug-in configuration performed after installing the plug-in.

2. Select Project -> Reconfigure Project. This runs the `autogen.sh` in the workspace for your project. The script runs `libtoolize`, `aclocal`, `autoconf`, `autoheader`, `automake` □□a, and `./configure`.

4.4. Building the Project

To build the project, select Project -> Build Project. You should see the console updated and you can note the cross-compiler you are using.

4.5. Starting QEMU in User Space NFS Mode

To start the QEMU emulator from within Eclipse, follow these steps:

1. Select Run -> External Tools -> External Tools Configurations... This selection brings up the External Tools Configurations Dialogue.

2. Go to the left navigation area and expand ‘Program’. You should find the image listed. For example, qemu-x86_64-poky-linux.

3. Click on the image. This brings up a new environment in the main area of the External Tools Configurations Dialogue. The Main tab is selected.

4. Click “Run” next. This brings up a shell window.

5. Enter your host root password in the shell window at the prompt. This sets up a Tap 0 connection needed for running in user-space NFS mode.

6. Wait for QEMU to launch.

7. Once QEMU launches you need to determine the IP Address for the user-space NFS. You can do that by going to a terminal in the QEMU and entering the `ipconfig` command.

4.6. Deploying and Debugging the Application

Once QEMU is running you can deploy your application and use the emulator to perform debugging. Follow these steps to deploy the application.

1. Select Run -> Debug Configurations...

2. In the left area expand “C/C++Remote Application”.

3. Locate your project and select it to bring up a new tabbed view in the Debug Configurations dialogue.

4. Enter the absolute path into which you want to deploy the application. Use the Remote Absolute File Path for C/C++Application:. For example, enter `/usr/bin/<programname>`.

5. Click on the Debugger tab to see the cross-tool debugger you are using.

6. Create a new connection to the QEMU instance by clicking on “new”.

7. Select “TCF, which means Target Communication Framework.

8. Click “Next”.

9. Clear out the “host name” field and enter the IP Address determined earlier.

10Click Finish to close the new connections dialogue.

11Use the drop-down menu now in the “Connection” field and pick the IP Address you entered.

12Click “Debug” to bring up a login screen and login.

13Accept the debug perspective.

4.7. Running User-Space Tools

As mentioned earlier in the manual several tools exist that enhance your development experience. These tools are aids in developing and debugging applications and images. You can run these user-space tools from within the Yocto Eclipse Plug-in through the Window -> YoctoTools menu.

Once you pick a tool you need to configure it for the remote target. Every tool needs to have the connection configured. You must select an existing TCF-based RSE connection to the remote target. If one does not exist, click "New" to create one.

Here are some specifics about the remote tools:

- OProfile: Selecting this tool causes the oprofile-server on the remote target to launch on the local host machine. The oprofile-viewer must be installed on the local host machine and the oprofile-server must be installed on the remote target, respectively, in order to use. You can locate both the viewer and server from http://git.yoctoproject.org/cgit/cgit.cgi/oprofileui/. You need to compile and install the oprofile-viewer from the source code on your local host machine. The oprofile-server is installed by default in the image.

- Lttng-ust: Selecting this tool runs "usttrace" on the remote target, transfers the output data back to the local host machine and uses "lttv-gui" to graphically display the output. The "lttv-gui" must be installed on the local host machine to use this tool. For information on how to use "lttng" to trace an application, see http://lttng.org/files/ust/manual/ust.html.

  For "Application" you must supply the absolute path name of the application to be traced by user mode lttng. For example, typing `/path/to/foo` triggers `usttrace /path/to/foo` on the remote target to trace the program `/path/to/foo`.

  "Argument" is passed to `usttrace` running on the remote target.

- PowerTOP: Selecting this tool runs "PowerTOP" on the remote target machine and displays the results in a new view called "powertop".

  "Time to gather data(sec):" is the time passed in seconds before data is gathered from the remote target for analysis.

  "show pids in wakeups list:" corresponds to the -p argument passed to "powertop".

- LatencyTOP and Perf: "LatencyTOP" identifies system latency, while "perf" monitors the system's performance counter registers. Selecting either of these tools causes an RSE terminal view to appear from which you can run the tools. Both tools refresh the entire screen to display results while they run.

Chapter 5. Using the Command Line

Recall that earlier we talked about how to use an existing toolchain tarball that had been installed into `/opt/poky`, which is outside of the Poky build environment (see Section 2.1.2, “Using an Existing Toolchain Tarball”. And, that sourcing your architecture-specific environment setup script initializes a suitable development environment. This setup occurs by adding the compiler, QEMU scripts, QEMU binary, a special version of `pkgconfig` and other useful utilities to the `PATH` variable. 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. These conditions allow you to easily use the toolchain outside of the Poky build environment on both autotools-based projects and makefile-based projects.

5.1. Autotools-Based Projects

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 in `/opt/poky` resulting from unpacking the cross-toolchain tarball. For example, the host option for an ARM-based target that uses the GNU EABI is `armv5te-poky-linux-gnueabi`. Note that the name of the script is `environment-setup-armv5te-poky-linux-gnueabi`. Thus, the following command works:

```
$ configure □□host-armv5te-poky-linux-gnueabi □□with-libtool-sysroot=<sysroot-dir>
```

This single command updates your project and rebuilds it using the appropriate cross-toolchain tools.

5.2. Makefile-Based Projects

For a makefile-based project you use the cross-toolchain by making sure the tools are used. You can do this as follows:

```
CC=arm-poky-linux-gnueabi-gcc
LD=arm-poky-linux-gnueabi-ld
CFLAGS=”${CFLAGS} □□sysroot=<sysroot-dir>”
CXXFLAGS=”${CXXFLAGS} □□sysroot=<sysroot-dir>”
```