The Yocto Project Application Development Toolkit (ADT) User's Guide

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

Note

Due to production processes, there could be differences between the Yocto Project documentation bundled in the release tarball and the Application Developer's Toolkit (ADT) User's Guide [http://www.yoctoproject.org/docs/1.1.1/adt-manual/adt-manual.html] on the Yocto Project [http://www.yoctoproject.org] website. For the latest version of this manual, see the manual on the website.

Table of Contents

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 Yocto Project build system Poky. The toolchain and sysroot are based on a metadata configuration and extensions, which allows you to cross-develop on the host machine for the target.

Additionally, to provide an effective development platform, the Yocto Project makes available and suggests other tools you can use with 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.

1.2. ADT Components

This section provides a brief description of what comprises the ADT.

1.2.1. The Cross-Toolchain

The cross-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 or through a Yocto Project build tree that is based on your metadata configuration or extension for your targeted device. The cross-toolchain works with a matching target sysroot.

1.2.2. 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 Yocto Project's build system Poky and uses the same metadata configuration used to build the cross-toolchain.

1.2.3. 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 downloaded a Yocto Project release and unpacked it to create a Yocto Project file structure and you have sourced the Yocto Project environment setup script, QEMU is installed and automatically available.
- If you have installed the cross-toolchain tarball and you have sourcing the toolchain's setup environment script, QEMU is also installed and automatically available.

1.2.4. 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/ 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, you must install it, `source` a script to set up the environment, and be sure both the kernel and filesystem image specific to the target architecture exist. This chapter describes how to be sure you meet the ADT requirements.

2.1. Installing the ADT

The following list describes how you can install the ADT, which includes the cross-toolchain. Regardless of the installation you choose, you must `source` the cross-toolchain environment setup script before you use the toolchain. See the "Setting Up the Cross-Development Environment" section for more information.

- 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 Tarball: Using this method, you select and download an architecture-specific toolchain tarball and then 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 a Yocto Project Build Tree: If you already have a Yocto Project build tree, you can build the cross-toolchain within tree. 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.

2.1.1. Using the ADT Installer

To run the ADT Installer, you need to first get the ADT Installer tarball and then run the ADT Installer Script.

2.1.1.1. Getting the ADT Installer Tarball

The ADT Installer is contained in the ADT Installer tarball. You can download the tarball into any directory from the Index of Releases [http://downloads.yoctoproject.org/releases], specifically at http://downloads.yoctoproject.org/releases/yocto/yocto-1.1.1/adt_installer. Or, you can use BitBake to generate the tarball inside the existing Yocto Project build tree.

If you use BitBake to generate the ADT Installer tarball, you must `source` the Yocto Project environment setup script (`oe-init-build-env`) located in the Yocto Project file structure before running the `bitbake` command that creates the tarball.

The following example commands download the Yocto Project release tarball, set up the Yocto Project files structure, set up the environment while also creating the default Yocto Project build tree, and run the `bitbake` command that results in the tarball `~/yocto-project/build/tmp/deploy/sdk/adt_installer.tar.bz2`:

```
     $ cd ~
     $ mkdir yocto-project
     $ cd yocto-project
     $ wget http://downloads.yoctoproject.org/releases/yocto/yocto-1.1.1/poky-edison-6.0.1.tar.bz
     $ tar xjf poky-edison-6.0.1.tar.bz2
     $ source poky-edison-6.0.1/oe-init-build-env
     $ bitbake adt-installer
```

2.1.1.2. 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 ~/yocto-project/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 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. 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 for this example using the following commands:

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

Note

The ADT Installer requires the `libtool` package to complete. If you install the recommended packages as described in the "Packages [http://www.yoctoproject.org/docs/1.1.1/yocto-project-qs/yocto-project-qs.html#packages]" section of The Yocto Project Quick Start [http://

www.yoctoproject.org/docs/1.1.1/yocto-project-qs/yocto-project-qs.html], then you will have libtool installed.

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, 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. You will notice environment setup files for the cross-toolchain in `/opt/poky/1.1.1`, 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.

2.1.2. Using a Cross-Toolchain Tarball

If you want to simply install the cross-toolchain by hand, you can do so by using an existing cross-toolchain tarball. If you use this method to install the cross-toolchain and you still need to install the target sysroot, you will have to install sysroot separately.

Follow these steps:

1. Go to http://downloads.yoctoproject.org/releases/yocto/yocto-1.1.1/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).

2. Go into that folder and download the toolchain tarball whose name includes the appropriate target architecture. For example, if your host development system is an Intel-based 64-bit system and you are going to use your cross-toolchain for an Intel-based 32-bit target, go into the `x86_64` folder and download the following tarball:

   ```
   poky-eglibc-x86_64-i586-toolchain-1.1.1.tar.bz2
   ```

   #### Note

   As an alternative to steps one and two, you can build the toolchain tarball if you have a Yocto Project build tree. If you need GMAE, you should use the `bitbake meta-toolchain-gmae` command. The resulting tarball will support such development. However, if you are not concerned with GMAE, you can generate the tarball using `bitbake meta-toolchain`.

   Use the appropriate `bitbake` command only after you have sourced the `oe-build-init-env` script located in the Yocto Project files. When the `bitbake` command completes, the tarball will be in `tmp/deploy/sdk` in the Yocto Project build tree.

3. Make sure you are in the root directory with root privileges and then expand the tarball. The tarball expands into `/opt/poky/1.1.1`. Once the tarball is expanded, the cross-toolchain is installed. You will notice environment setup files for the cross-toolchain in the directory. Here is an example where the tarball exists in the user's `Downloads` directory:

   ```
   # cd /
   # tar -xjf /home/scottrif/Downloads/poky-eglibc-x86_64-i586-toolchain-gmae-1.1.tar.bz2
   ```

2.1.3. Using BitBake and the Yocto Project Build Tree

A final way of installing just the cross-toolchain is to use BitBake to build the toolchain within an existing Yocto Project build tree. This method does not install the toolchain into the `/opt` directory. As with the previous method, if you need to install the target sysroot, you must do this separately.

Follow these steps to build and install the toolchain into the build tree:

1. Source the environment setup script `oe-init-build-env` located in the Yocto Project files.

2. At this point, you should be sure that the `MACHINE` variable in the `local.conf` file found in the `conf` directory of the Yocto Project 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 tree 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. Run `bitbake meta-ide-support` to complete the cross-toolchain installation.

> **Note**
>
> If you change out of your working directory after you `source` the environment setup script and before you run the `bitbake` command, the command might not work. Be sure to run the `bitbake` command immediately after checking or editing the `local.conf` but without changing out of your working directory.

   Once the `bitbake` command finishes, the tarball for the cross-toolchain is generated within the Yocto Project build tree. You will notice environment setup files for the cross-toolchain in the Yocto Project build tree in the `tmp` directory. Setup script filenames contain the strings `environment-setup`.

2.2. 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 used an existing ADT tarball to install the ADT, then you can find this script in the `/opt/poky/1.1.1` directory. If you installed the toolchain in the build tree, you can find the environment setup script for the toolchain in the Yocto Project build tree's `tmp` directory.

Be sure to source 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 command to source the toolchain environment setup script for a 64-bit IA-based machine would be the following:

```
     $ source /opt/poky/1.1.1/environment-setup-x86_64-poky-linux
```

2.3. 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.

2.3.1. 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/1.1.1/yocto-project-qs/yocto-project-qs.html#test-run]" section of The Yocto Project Quick Start.

The Yocto Project provides 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 Yocto Project release area - http://downloads.yoctoproject.org/releases/yocto/yocto-1.1.1/machines/ and are ideal for experimentation within Yocto Project. For information on the image types you can build using the Yocto Project, see the "Reference: Images [http://www.yoctoproject.org/docs/1.1.1/poky-ref-manual/poky-ref-manual.html#ref-images]" appendix in The Yocto Project Reference Manual.

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 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 file in the Yocto Project build directory and then rebuild the image. With this method, you need to modify the `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" 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, which is stored using the Subversion SCM, using the following command:

     ```
     $ svn checkout svn://dev.eclipse.org/svnroot/dsdp/org.eclipse.tm.tcf/trunk/agent \
        <-r #rev_number>
     ```

  4. Modify the `Makefile.inc` file for the cross-compilation environment by setting the `OPSYS` and `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.

2.3.2. Extracting the Root Filesystem

You must extract the root filesystem if you want to boot the image using NFS or 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. Another example is if 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 and then use the `runqemu-extract-sdk` command on the filesystem image tarball. For example, the following commands set up the environment by sourcing the setup script from within the build directory and then extracting the root filesystem from a previously built filesystem image tarball named `core-image-sato-sdk-qemux86.tar.bz2`. The example extracts the root filesystem into the `$HOME/qemux86-sato` directory:

```
$ source $HOME/poky/build/tmp/environment-setup-i586-poky-linux
$ runqemu-extract-sdk \
   tmp/deploy/images/core-image-sato-sdk-qemux86.tar.bz2 \
   $HOME/qemux86-sato
```

In this case, you could now point to the target sysroot at `$HOME/qemux86-sato`.

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 target sysroot.

3.1. Package Management Systems

The Yocto Project 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 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 at sysroot. Additional values specify additional formats for convenience or testing. See the configuration file for details.

Note

For build performance information related to the PMS, see Packaging - `package*.bbclass` [http://www.yoctoproject.org/docs/1.1.1/poky-ref-manual/poky-ref-manual.html#ref-classes-package] 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 environment setup script found in the Yocto Project files. 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 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, 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.

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. Install the optimal version of the Eclipse IDE.

2. Configure the Eclipse IDE.

3. Install the Eclipse Yocto Plug-in.

4. Configure the Eclipse Yocto Plug-in.

4.1.1. Installing the Eclipse IDE

It is recommended that you have the Indigo 3.7 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 particular to your development host. 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. For example, the following commands unpack and install the Eclipse IDE tarball found in the `Downloads` area into a clean directory using the default name `eclipse`:

```
     $ cd ~
     $ tar -xzvf ~/Downloads/eclipse-SDK-3.7.1-linux-gtk-x86_64.tar.gz
```

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 metadata 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. Configuring the Eclipse IDE

Before installing and configuring the Eclipse Yocto Plug-in, you need to configure the Eclipse IDE. Follow these general steps to configure Eclipse:

1. Start the Eclipse IDE.
2. Make sure you are in your Workbench and select "Install New Software" from the "Help" pull-down menu.
3. Select `indigo - http://download.eclipse.org/releases/indigo` from the "Work with:" pull-down menu.
4. Expand the box next to `Programming Languages` and select the `Autotools Support for CDT (incubation)` and `C/C++ Development Tools` boxes.
5. Complete the installation and restart the Eclipse IDE.
6. After the Eclipse IDE restarts and from the Workbench, select "Install New Software" from the "Help" pull-down menu.
7. Click the "Available Software Sites" link.
8. Check the box next to `http://download.eclipse.org/tm/updates/3.3` and click "OK".
9. Select `http://download.eclipse.org/tm/updates/3.3` from the "Work with:" pull-down menu.
10. Check the box next to `TM and RSE Main Features`.
11. Expand the box next to `TM and RSE Optional Add-ons` and select every item except `RSE Unit Tests` and `RSE WinCE Services (incubation)`.
12. Complete the installation and restart the Eclipse IDE.
13. If necessary, select "Install New Software" from the "Help" pull-down menu so you can click the "Available Software Sites" link again.
14. After clicking "Available Software Sites", check the box next to `http://download.eclipse.org/tools/cdt/releases/indigo` and click "OK".
15. Select `http://download.eclipse.org/tools/cdt/releases/indigo` from the "Work with:" pull-down menu.
16. Check the box next to `CDT Main Features`.
17. Expand the box next to `CDT Optional Features` and select `C/C++ Remote Launch` and `Target Communication Framework (incubation)`.
18. Complete the installation and restart the Eclipse IDE.

4.1.3. Installing or Accessing the Eclipse Yocto Plug-in

You can install the Eclipse Yocto Plug-in into the Eclipse application one of two ways: using the Eclipse IDE and installing the plug-in as new software, or using a built zip file. If you don't want to permanently install the plug-in but just want to try it out within the Eclipse environment, you can import the plug-in project from the Yocto Project source repositories.

4.1.3.1. Installing the Plug-in as New Software

To install the Eclipse Yocto Plug-in as new software directly into the Eclipse IDE, follow these steps:

1. Start up the Eclipse IDE.
2. In Eclipse, select "Install New Software" from the "Help" menu.
3. Click "Add..." in the "Work with:" area.
4. Enter `http://downloads.yoctoproject.org/releases/eclipse-plugin/1.1.1` in the URL field and provide a meaningful name in the "Name" field.
5. Click "OK" to have the entry added to the "Work with:" drop-down list.
6. Select the entry for the plug-in from the "Work with:" drop-down list.

7. Check the box next to `Development tools and SDKs for Yocto Linux`.
8. Complete the remaining software installation steps and then restart the Eclipse IDE to finish the installation of the plug-in.

4.1.3.2. Installing the Plug-in from a Zip File

To install the Eclipse Yocto Plug-in by building and installing a plug-in zip file, follow these steps:

1. Open a shell and create a Git repository with:

   ```
   $ git clone git://git.yoctoproject.org/eclipse-poky yocto-eclipse
   ```

   For this example, the repository is named `~/yocto-eclipse`.
2. Locate the `build.sh` script in the Git repository you created in the previous step. The script is located in the `scripts`.
3. Be sure to set and export the `ECLIPSE_HOME` environment variable to the top-level directory in which you installed the Indigo version of Eclipse. For example, if your Eclipse directory is `$HOME/eclipse`, use the following:

   ```
   $ export ECLIPSE_HOME=$HOME/eclipse
   ```
4. Run the `build.sh` script and provide the name of the Git branch along with the Yocto Project release you are using. Here is an example that uses the `master` Git repository and the `1.1.1` release:

   ```
   $ scripts/build.sh master 1.1.1
   ```

   After running the script, the file `org.yocto.sdk-<release>-<date>-archive.zip` is in the current directory.
5. If necessary, start the Eclipse IDE and be sure you are in the Workbench.
6. Select "Install New Software" from the "Help" pull-down menu.
7. Click "Add".
8. Provide anything you want in the "Name" field.
9. Click "Archive" and browse to the ZIP file you built in step four. This ZIP file should not be "unzipped", and must be the `*archive.zip` file created by running the `build.sh` script.
10. Check the box next to the new entry in the installation window and complete the installation.
11. Restart the Eclipse IDE if necessary.

At this point you should be able to configure the Eclipse Yocto Plug-in as described in the "Configuring the Eclipse Yocto Plug-in" section.

4.1.3.3. Importing the Plug-in Project into the Eclipse Environment

Importing the Eclipse Yocto Plug-in project from the Yocto Project source repositories is useful when you want to try out the latest plug-in from the tip of plug-in's development tree. It is important to understand when you import the plug-in you are not installing it into the Eclipse application. Rather, you are importing the project and just using it. To import the plug-in project, follow these steps:

1. Open a shell and create a Git repository with:

   ```
   $ git clone git://git.yoctoproject.org/eclipse-poky yocto-eclipse
   ```

For this example, the repository is named `~/yocto-eclipse`.

2. In Eclipse, select "Import" from the "File" menu.

3. Expand the "General" box and select "existing projects into workspace" and then click "Next".

4. Select the root directory and browse to "~/yocto-eclipse/plugins".

5. There will be three things there. Select each one and install one at a time. Do all three.

The left navigation pane in the Eclipse application shows the default projects. Right-click on one of these projects and run it as an Eclipse application. This brings up a second instance of Eclipse IDE that has the Yocto Plug-in.

4.1.4. Configuring the Eclipse Yocto Plug-in

Configuring the Eclipse Yocto Plug-in involves setting the Cross Compiler options and the Target options. The configurations you choose become the default settings for all projects. You do have opportunities to change them later when you configure the project (see the following section).

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

- Choose `Windows -> Preferences` to display the `Preferences` Dialog
- Click `Yocto ADT`

4.1.4.1. Configuring the Cross-Compiler Options

To configure the Cross Compiler Options, you must select the type of toolchain, point to the toolchain, specify the sysroot location, and select the target architecture.

- Selecting the Toolchain Type: Choose between `Standalone pre-built toolchain` and `Build system derived toolchain` for Cross Compiler Options.
  - `Standalone Pre-built Toolchain`: Select this mode when you are using a stand-alone cross-toolchain. For example, suppose you are an application developer and do not need to build a target image. Instead, you just want to use an architecture-specific toolchain on an existing kernel and target root filesystem.
  - `Build System Derived Toolchain`: Select this mode if the cross-toolchain has been installed and built as part of the Yocto Project build tree. When you select `Build system derived toolchain`, you are using the toolchain bundled inside the Yocto Project build tree.
- Point to the Toolchain: If you are using a stand-alone pre-built toolchain, you should be pointing to the `/opt/poky/1.1.1` directory. This is the location for toolchains installed by the ADT Installer or by hand. Sections "Configuring and Running the ADT Installer Script" and "Using a Cross-Toolchain Tarball" describe two ways to install a stand-alone cross-toolchain in the `/opt/poky` directory.

  Note

  It is possible to install a stand-alone cross-toolchain in a directory other than `/opt/poky`. However, doing so is discouraged.

  If you are using a system-derived toolchain, the path you provide for the `Toolchain Root Location` field is the Yocto Project's build directory. See section "Using BitBake and the Yocto Project Build Tree" for information on how to install the toolchain into the Yocto Project build tree.
- Specify the Sysroot Location: This location is where the root filesystem for the target hardware is created on the development system by the ADT Installer. The QEMU user-space tools, the NFS boot process, and the cross-toolchain all use the sysroot location.
- Select the Target Architecture: The target architecture is the type of hardware you are going to use or emulate. Use the pull-down `Target Architecture` menu to make your selection. The pull-down menu should have the supported architectures. If the architecture you need is not listed in the menu, you will need to build the image. See the "Building an Image [http://www.yoctoproject.org/docs/1.1.1/yocto-project-qs/yocto-project-qs.html#building-image]" section of The Yocto Project Quick Start for more information.

4.1.4.2. Configuring the Target Options

You can choose to emulate hardware using the QEMU emulator, or you can choose to run your image on 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 specify any custom options.

  If you selected `Build system derived toolchain`, the target kernel you built will be located in the Yocto Project build tree in `tmp/deploy/images` directory. If you selected `Standalone pre-built toolchain`, the pre-built image 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 as part of the Cross Compiler Options configuration in the `Sysroot Location:` field.

- `External HW:` Select this option if you will be using actual hardware.

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 Makefile-based projects in a terminal window, see the section "Using the Command Line".

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. Expand `Yocto ADT 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. Do not use hyphens as part of the name.
7. Click `Next`.
8. Add information in the `Author` and `Copyright notice` fields.
9. Be sure the `License` field is correct.
10. Click `Finish`.
11. If the "open perspective" prompt appears, click "Yes" so that you in the C/C++ perspective.
12. The left-hand navigation pane shows your project. You can display your source by double clicking the project's source file.

4.3. Configuring the Cross-Toolchains

The earlier section, "Configuring the Eclipse Yocto Plug-in", sets up the default project configurations. You can override these settings for a given project by following these steps:

1. Select `Project -> Change Yocto Project Settings`: This selection brings up the `Project Yocto Settings` Dialog and allows you to make changes specific to an individual project.

   By default, the Cross Compiler Options and Target Options for a project are inherited from settings you provide using the `Preferences` Dialog as described earlier in the "Configuring the Eclipse Yocto Plug-in" section. The `Project Yocto Settings` Dialog allows you to override those default settings for a given project.

2. Make your configurations for the project and click "OK".

3. Select `Project -> Reconfigure Project`: This selection reconfigures the project by running `autogen.sh` in the workspace for your project. The script also runs `libtoolize`, `aclocal`, `autoconf`, `autoheader`, `automake --a`, and `./configure`. Click on the `Console` tab beneath your source code to see the results of reconfiguring your project.

4.4. Building the Project

To build the project, select `Project -> Build Project`. The console should update 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. Expose the `Run -> External Tools` menu. Your image should appear as a selectable menu item.

2. Select your image from the menu to launch the emulator in a new window.

3. If needed, 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.

4. Wait for QEMU to launch.

5. Once QEMU launches, you can begin operating within that environment. For example, you could determine the IP Address for the user-space NFS by using the `ifconfig` command.

4.6. Deploying and Debugging the Application

Once the QEMU emulator is running the image, using the Eclipse IDE 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` Dialog.

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

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

6. Click on the `Main` tab.

7. Create a new connection to the QEMU instance by clicking on new.

8. Select TCF, which means Target Communication Framework.

9. Click `Next`.

10. Clear out the `host name` field and enter the IP Address determined earlier.

11. Click `Finish` to close the `New Connections` Dialog.

12. Use the drop-down menu now in the `Connection` field and pick the IP Address you entered.

13. Click Debug to bring up a login screen and login.

14. Accept 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 Eclipse IDE through the `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 must compile and install the `oprofile-viewer` from the source code on your local host machine. Furthermore, in order to convert the target's sample format data into a form that the host can use, you must have `oprofile` version 0.9.4 or greater installed on the host.

  You can locate both the viewer and server from http://git.yoctoproject.org/cgit/cgit.cgi/oprofileui/.

  ### Note
  The `oprofile-server` is installed by default on the `core-image-sato-sdk` 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 the manual discussed how to use an existing toolchain tarball that had been installed into `/opt/poky`, which is outside of the Yocto Project build tree (see the section "Using an Existing Toolchain Tarball)". And, that sourcing your architecture-specific environment setup script initializes a suitable cross-toolchain development environment. During the setup, locations for the compiler, QEMU scripts, QEMU binary, a special version of `pkgconfig` and other useful utilities are added 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 Yocto Project 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.

Note

If `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 re-running the script:

```
$ 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 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>”
```