



1 This guide is useful for those who want to compile and debug Yocto modules and specifically OIP 2 modules. It's about creating a proper development image (including debug information and tracing tools), generating of SDK, installing of Eclipse CDT with the necessary plugins, environment configuration.

2 Setup: Development PC (Host) and device (Target) which runs Yocto based distribution. The point is that Target's file system(FS) is mapped to Host's file system. Developer uses cross-compilation environment to create new Yocto modules, which after compilation are just copied to Target's FS. Target's FS is mounted as NFS to Host. Using different tools provided by Eclipse CDT these modules can be debugged. As Target we're going to use QEmu based build, but the following steps apply the same way for real hardware.

- **Compilation of development cpi-image:**

We're going to use the current OIP 2.0 build as a base. In order to have a debug-able image a few tools needs to be added to the image. To do that we need to modify **local.conf** file. **Note that every time build environment is setup, local.conf contents are initialized to default!!!**

gedit <OIP 2.0 folder>/build/conf/local.conf

Replace **EXTRA_IMAGE_FEATURES = "debug-tweaks"**

with **EXTRA_IMAGE_FEATURES = "debug-tweaks tools-debug eclipse-debug tools-profile dbg-pkgs"**

In order to resolve licensing issues we need to comment the following line:

#INCOMPATIBLE_LICENSE ?= "GPLv3"

*And then we can recompile **cpi-image** by the usual way (after build environment has already been setup):*

bitbake cpi-image

The output image with all needed debug stuff resides at:

<OIP 2.0 folder>/build/tmp/deploy/images/qemux86-64

You also have to prepare a support for Eclipse CDT debugging:

bitbake meta-ide-support



Additional Debug Info

More info: https://wiki.yoctoproject.org/wiki/Tracing_and_Profiling#General_Setup



To skip compilation of all those images, they could be taken from somebody who has already did that, as building takes quite a lot of time.

• SDK creation:

bitbake cpi-image -c populate_sdk

The results is:

<OIP 2.0 folder>/build/tmp/deploy/sdk/oecore-x86_64-core2-64-toolchain-nodistro.0.sh

• SDK installation:

Create a new directory (E.g. <OIP 2.0 folder>/DEBUG)

```
cd <OIP 2.0 folder>/DEBUG
cp <OIP 2.0 folder>/build/tmp/deploy/sdk/oecore-x86_64-core2-64-toolchain-nodistro.0.sh ./
cp <OIP 2.0 folder>/build/tmp/deploy/images/qemux86-64/cpi-image-qemux86-64-<date>.rootfs.tar.bz2 ./
cp <OIP 2.0 folder>/build/tmp/deploy/images/qemux86-64/bzImage--3.xx.xx+git0+<code>-qemux86-64-<date>.bin ./
./oecore-x86_64-core2-64-toolchain-nodistro.0.sh -d ./oip-sdk/qemux86-64

# The last command install SDK to folder <OIP 2.0 folder>/DEBUG/oip-sdk/qemux86-64
```

• Extraction of Target's FS:

```
cd <OIP 2.0 folder>/DEBUG/oip-sdk/qemux86-64
source environment-setup-core2-64-poky-linux
runqemu-extract-sdk <OIP 2.0 folder>/DEBUG/cpi-image-qemux86-64-20160322132908.rootfs.tar.bz2 ./sysroots/oip2-platform
```

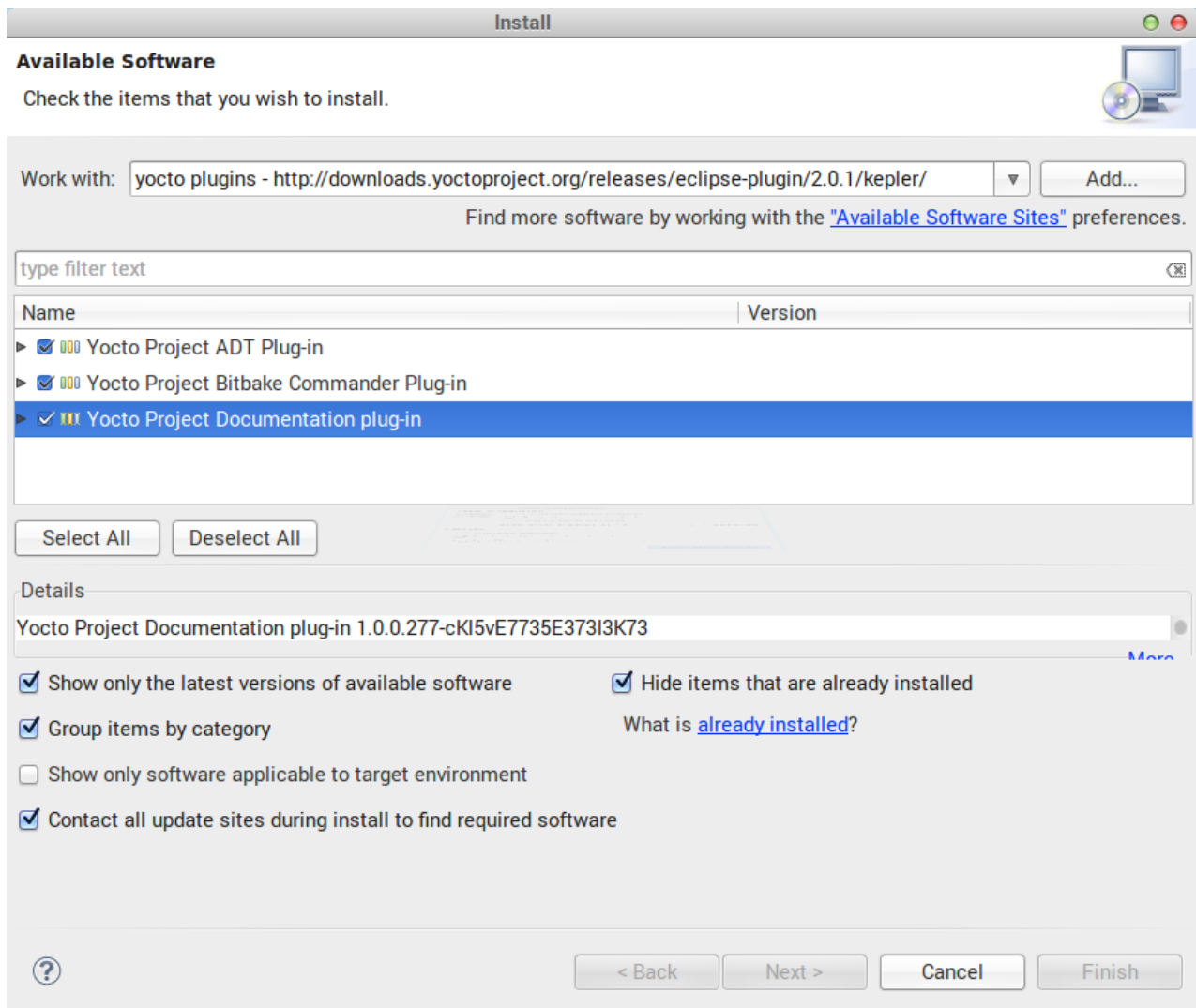
- **Installation of Eclipse CDT and Yocto plugins:**

**Supported Eclipse CDT version**

This manual uses Eclipse Kepler (4.3) as it is supported by Yocto. Newer releases doesn't seem to support the features we need e.g. Eclipse Mars.

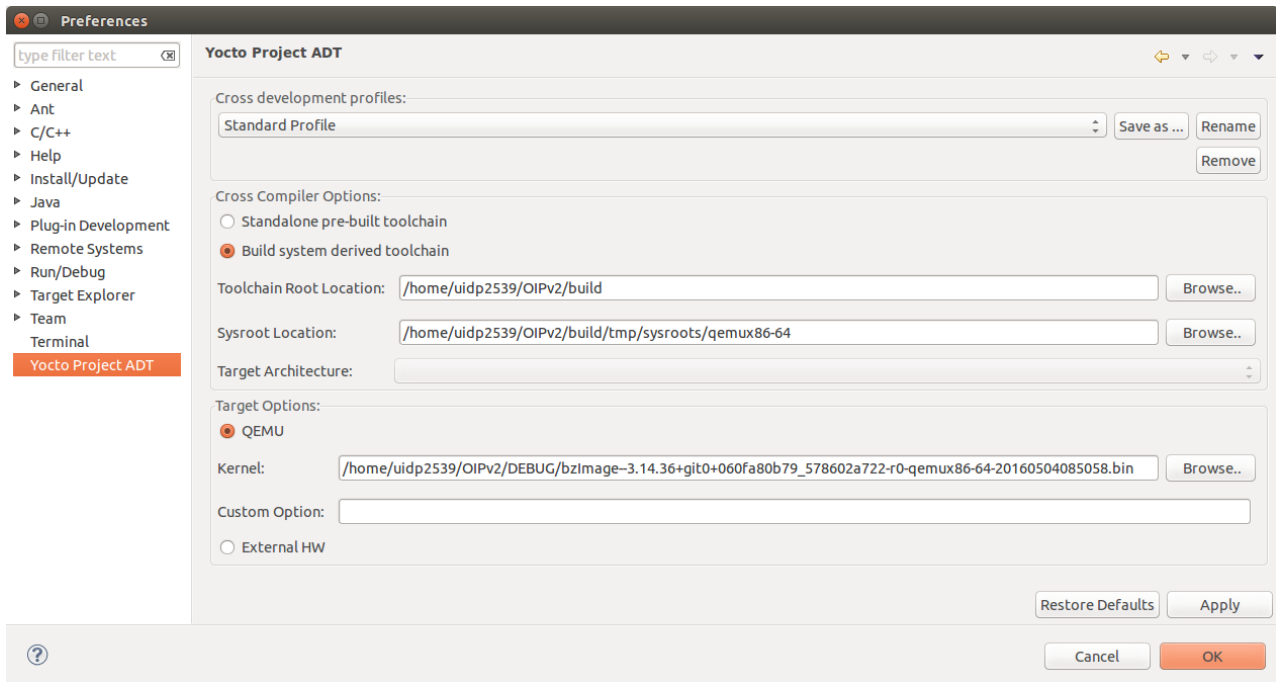
Eclipse requires Java Development Kit (JDK) to be installed. Note that if you want to install x64 version of Eclipse CDT, you'll also need x64 version of JDK and the same for 32bit versions.

1. Download and install [Eclipse Standart 4.3](#)
2. Start Eclipse and choose Help -> Install New Software, then from the "Work With" drop down menu choose "Kepler - <http://download.eclipse.org/releases/kepler>"
3. Install the following plugins:
 - From "Linux Tools" group:
"LTTng -Linux Tracing Toolkit"
 - From "Mobile and Device Development" group:
"C/C++ Remote Launch"
"Remote System Explorer End-user Runtime"
"Remote System Explorer User Actions"
"Target Management Terminal"
"TCF Remote System Explorer add-in"
"TCF Target Explorer"
4. Then again from Help -> Install New Software, then from the "Work With" drop down menu choose "<http://downloads.yoctoproject.org/releases/eclipse-plugin/2.0.1/kepler/>"



- **Setting up toolchain and sysroot in Eclipse**

From Windows -> Preferences choose Yocto Project ADT and fill **toolchain** and **sysroot** location as shown:



There are two possible configurations for selecting toolchain:

- 1) Standalone pre-build toolchain - this option uses the created SDK and then in **toolchain** field has to be selected the path to the SDK: <OIP 2.0 folder>/DEBUG/oip-sdk/qemux86-64
- 2) System derived toolchain - this option uses already setup environment by Yocto (as shown on the image).

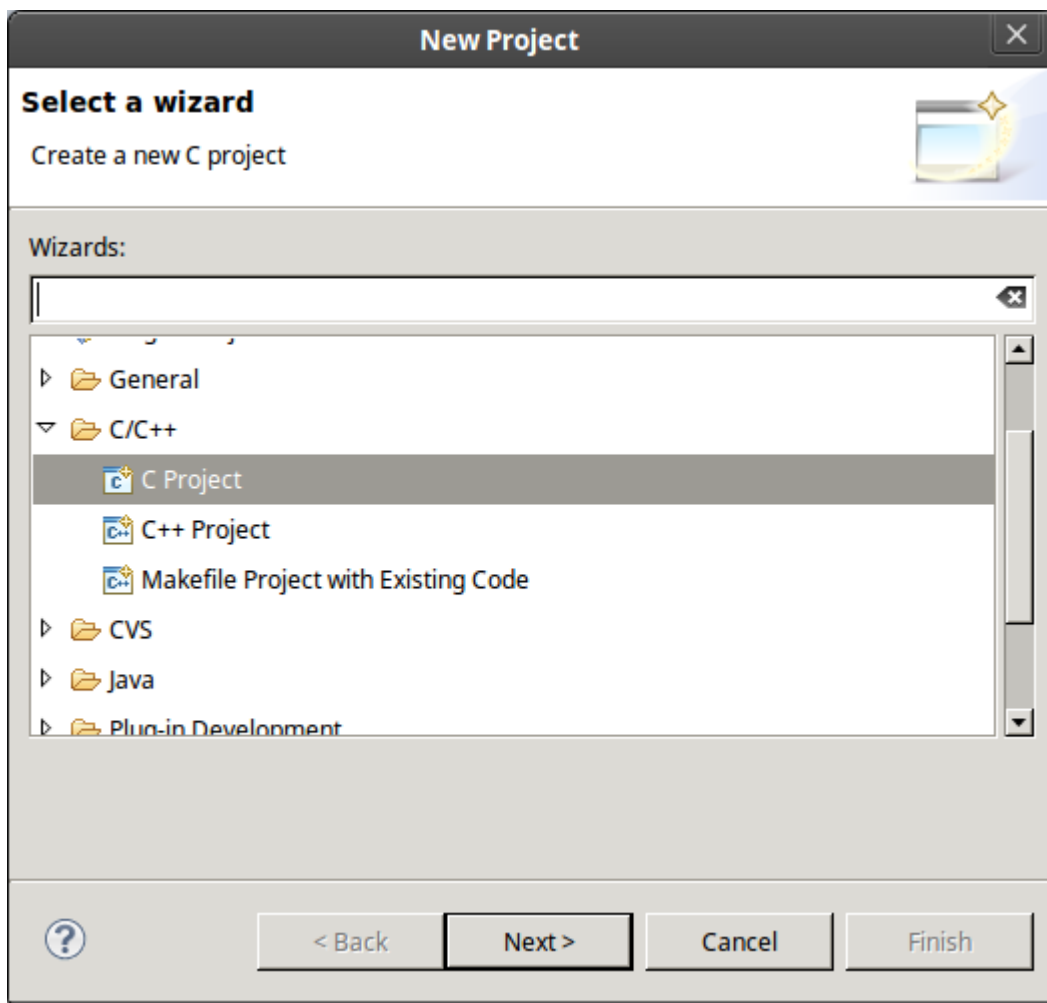


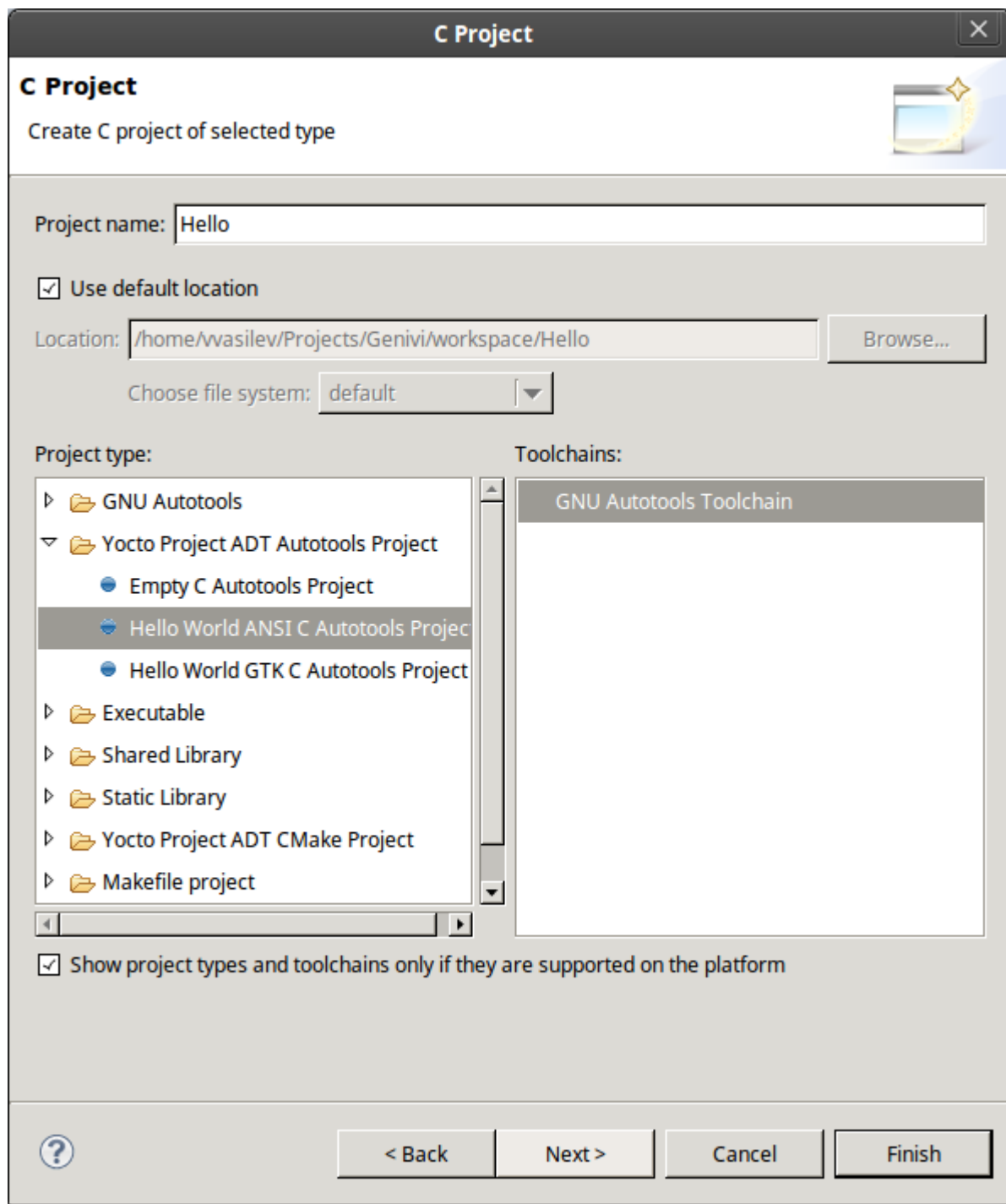
Options

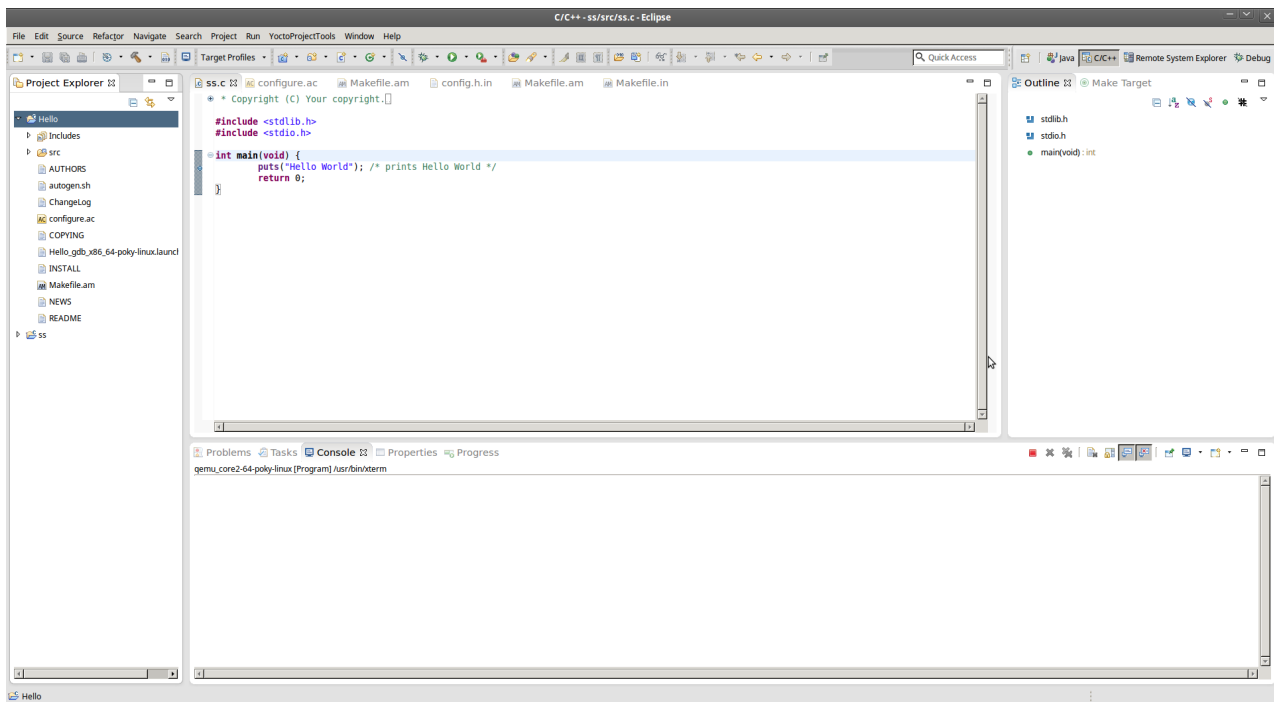
Currently only option 2 is tested and working

• Example project in Eclipse:

- 1) From File -> New -> Project choose "C Project"
- 2) Then from "Yocto Project ADT Autotools Project" select "Hello World ANSI C Autotools Project"







How to debug already present OIP 2.0 module

If you want to debug already present module, you need to create "Empty C Autotools project" and then in the created by Eclipse folder (usually in it's workspace) you have to copy/link the files to your module.

- **Compiling the test project:**

Select Project -> Build Project

- **Creating connection to the target**



- 1) Press "Open Perspective" button find and choose "Remote System Explorer"
- 2) "Remote Systems" tab appear on the left side.
- 3) Right-click and select "New Connection" and choose TCF
- 4) Fill the IP that your QEmu image is using. In our case 192.168.7.1
- 5) In Description field type QEmu
- 6) Before clicking Finish, start your QEmu with the debug cpi-image. For OIP 2.0 it is `~/OIPv2/build$./meta-sw-platform-oip/scripts/runqemu cpi-image`



Firewall

You must add the following rules for firewall configuration in order to have Eclipse CDT TCF connection with QEmu. Also you need to have rw FS so that Eclipse CDT can transfer the compiled binary to the target:

```
mount -o remount,rw /
```

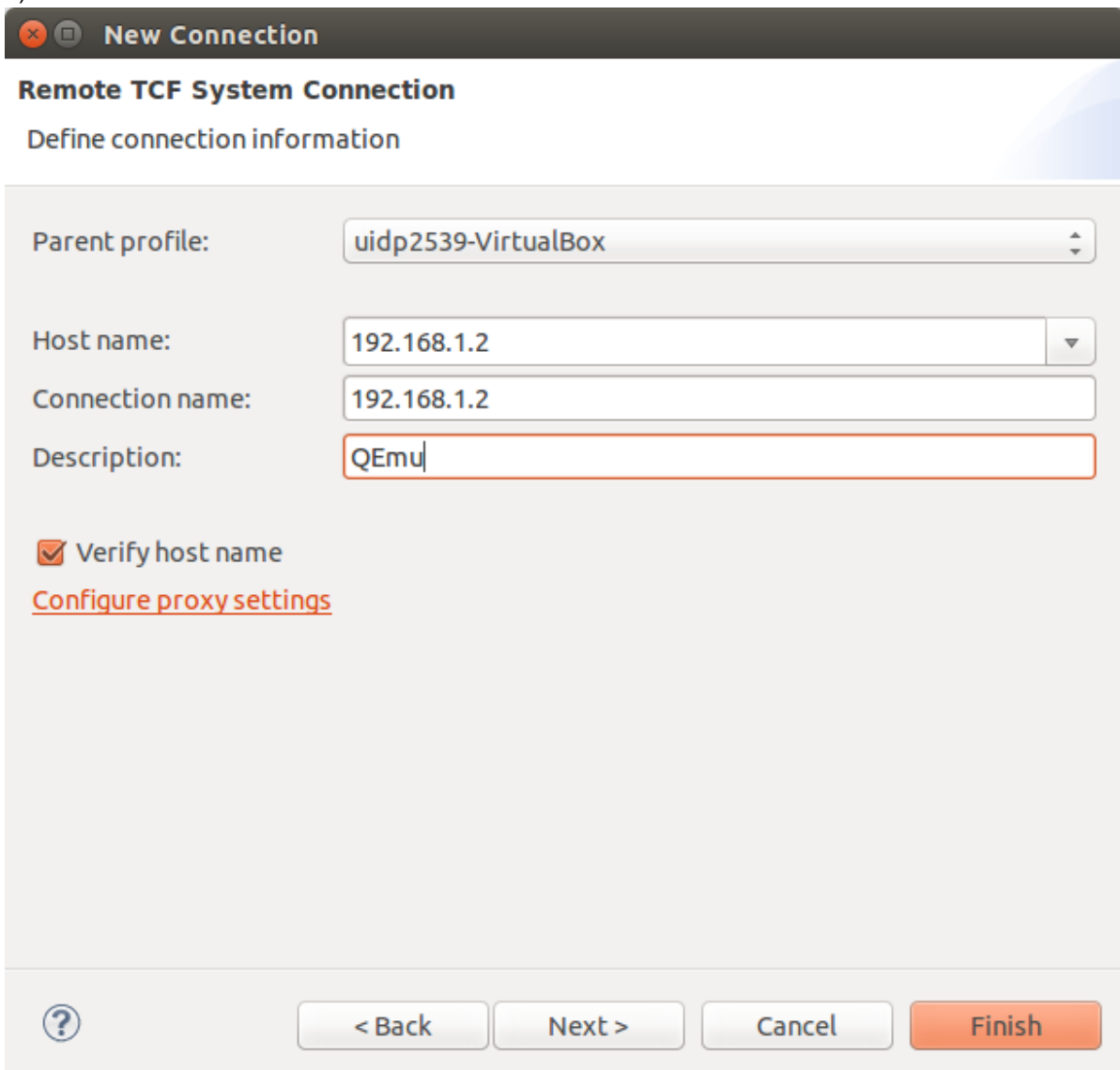
```
iptables -P INPUT ACCEPT
```

```
iptables -P OUTPUT ACCEPT
```

```
iptables -P FORWARD ACCEPT
```

These rules can be added in a script to the filesystem so they are easily applied every time.

7) Press Finish.



New Connection

Remote TCF System Connection

Define connection information

Parent profile: uidp2539-VirtualBox

Host name: 192.168.1.2

Connection name: 192.168.1.2

Description: QEmu


☒ Verify host name

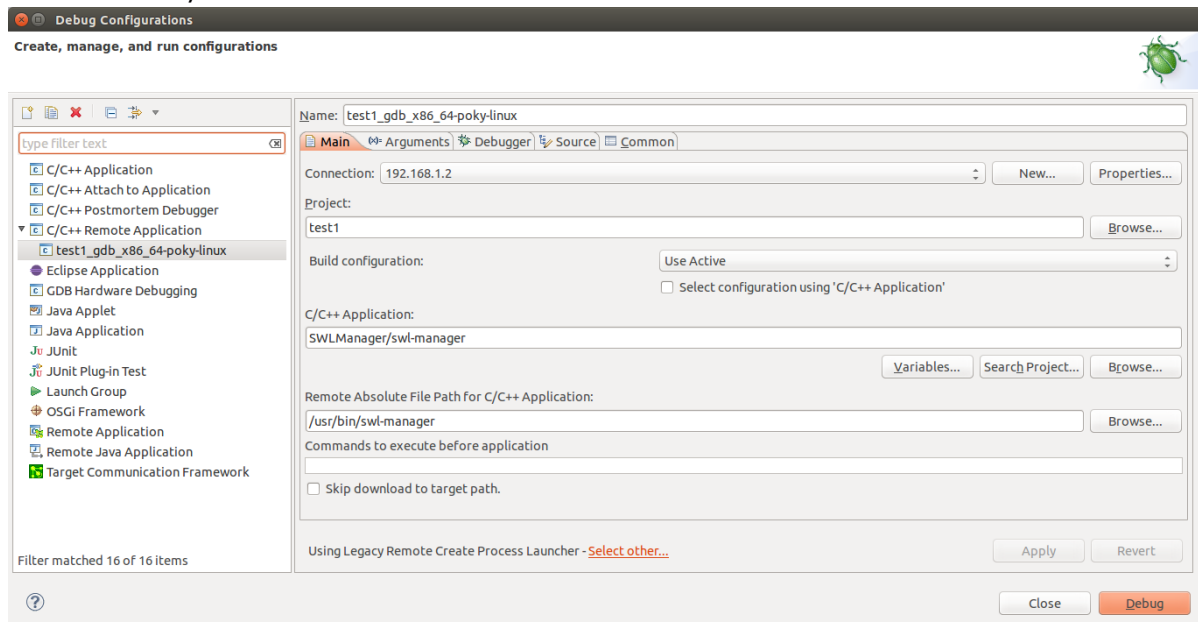
[Configure proxy settings](#)

? < Back Next > Cancel Finish

• Debug

After QEmu has been started we can initiate debug session for the compiled module.

Press the V icon next to Debug Button  and then choose "Debug Configurations". There's already a predefined configuration. *Make sure you have selected the new TCF connection in the Connection dropdown as shown below.*



*Press **DEBUG** button and session shall be initiated. Now you can debug your application as it is local.*