

# Steps to setup and debug generated Papyrus-RT models on Ubuntu 12.04 using CMake

## Contents

<b>OVERVIEW</b>	<b>2</b>
<b>REQUIREMENTS</b>	<b>2</b>
<b>STEPS</b>	<b>2</b>
1. Setup required directories	2
2. Create an external CMake tool	3
3. Generate the CMake structure	4
4. Verify the CDT Project's Toolchain	5
5. Update the Make Target	6
6. Build the CDT Project	7
7. Setup a Debug Configuration	8
8. Debug the Modeled Application	9

## Overview

This pictorial guide describes a basic configuration for building and debugging generated Papyrus-RT model projects. The developer should be familiar with generating Papyrus-RT model projects, and a basic understanding of CMake. This document does not address installation procedures nor provide methods for troubleshooting related code generation and build issues.

## Requirements

The steps below depend on the following Ubuntu 12.04 configuration:

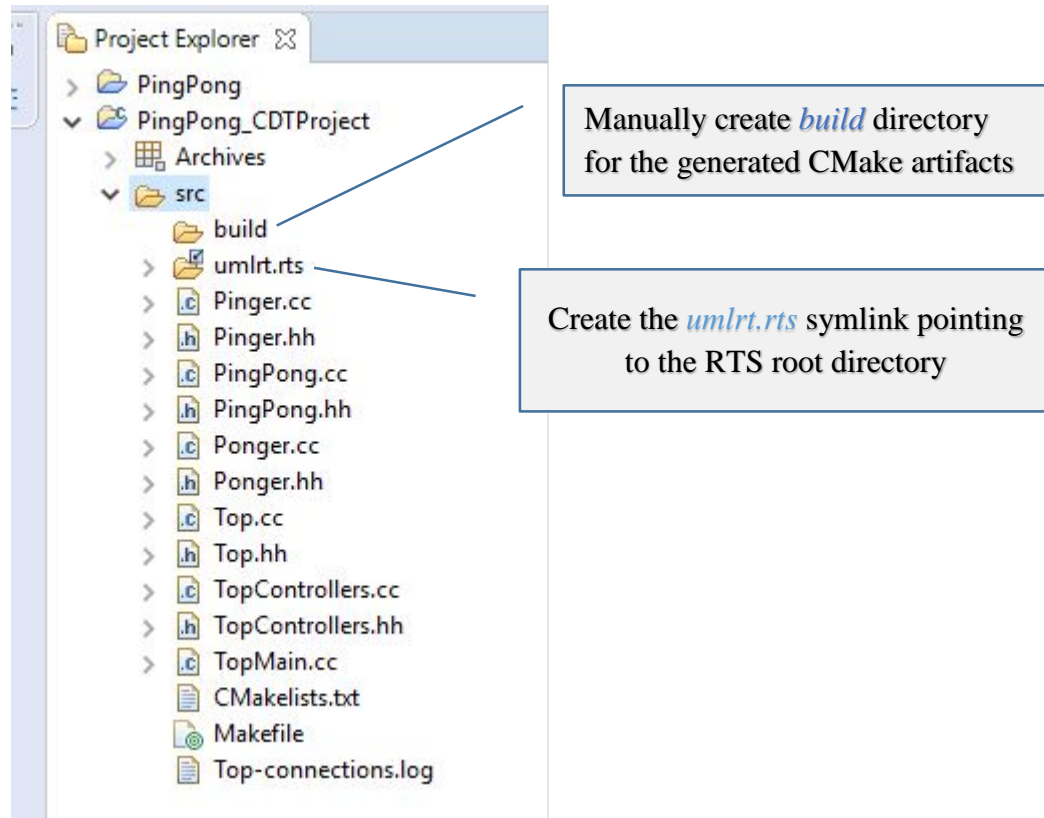
- The Neon developer environment for Papyrus-RT
- An installation of CMake and related dev packages
- The PingPong project tutorial

## Steps

### 1. Setup required directories

The generated CDT project requires an additional build directory, and a symlink to the RTS root directory. The additional *build* directory has no particular naming constraints. The symlink, however, should be named *umlrt.rts* to avoid the necessity of undocumented configuration changes. For example:

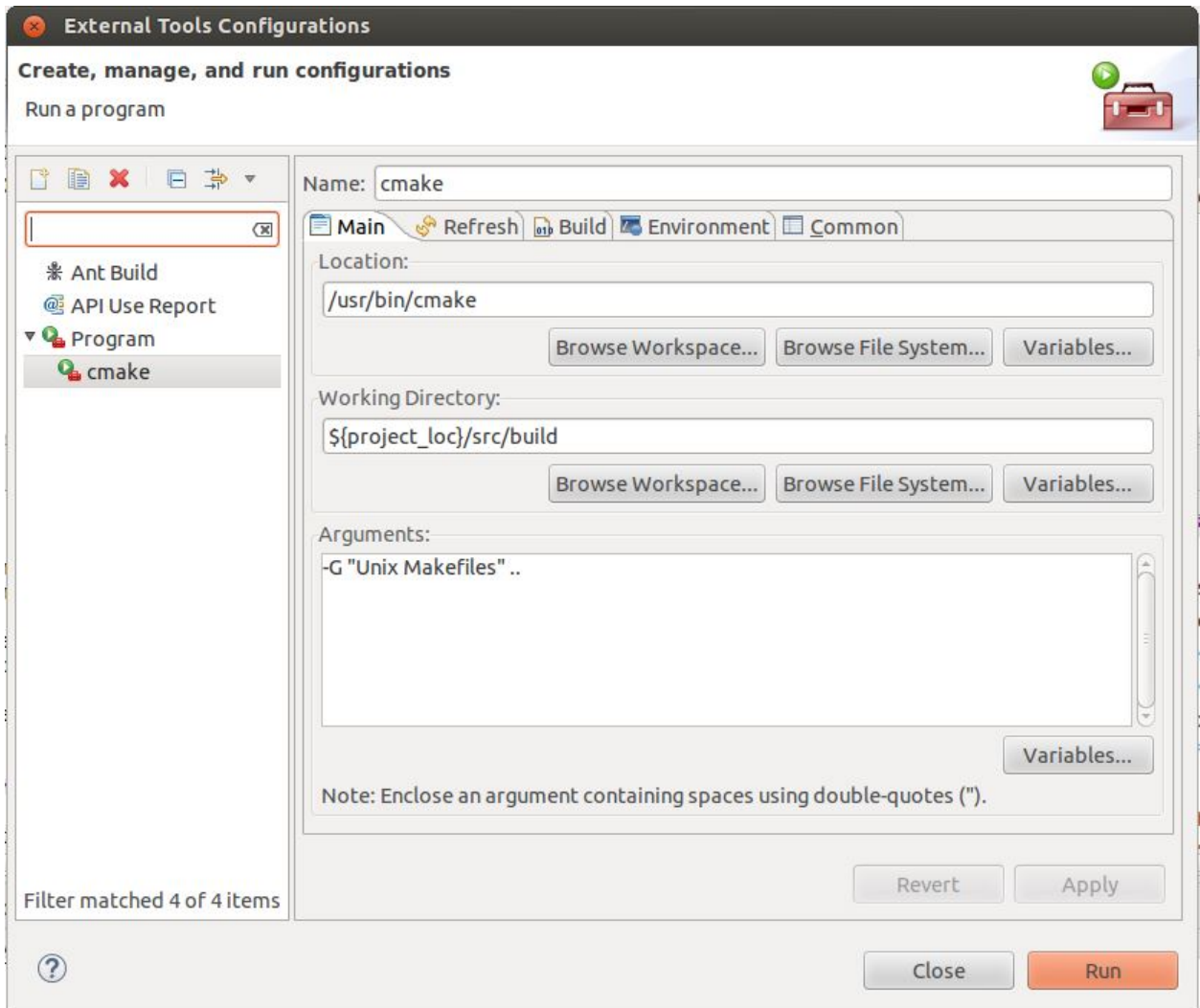
```
umlrt.rts -> ~/papyrus-rt-master/git/org.eclipse.papyrus-rt/plugins/umlrt/runtime/rts
```



## 2. Create an external CMake tool

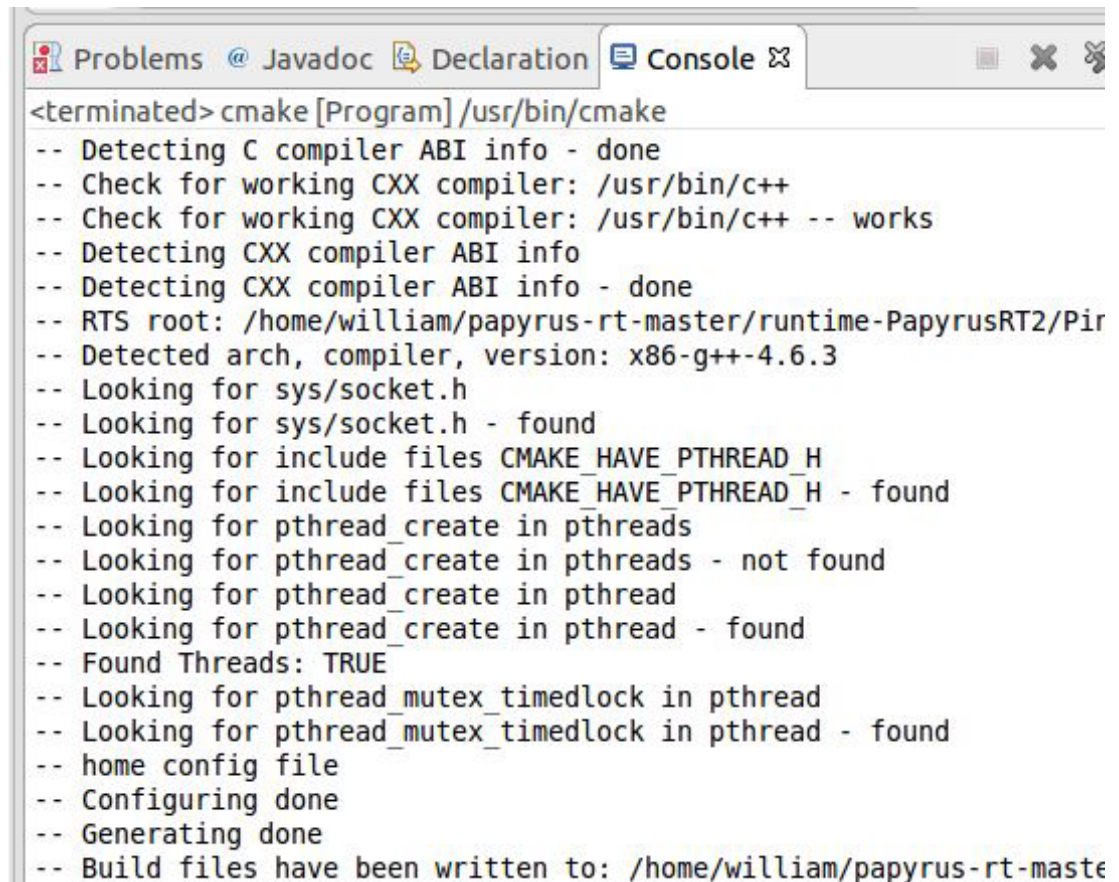
A CMake launch configuration is setup to execute from within the Eclipse IDE. The following tool configuration shows how this can be accomplished. The Working Directory specifies the build directory created in the previous step. For clarity, the CMake arguments:

`-G "Unix Makefiles" ..`



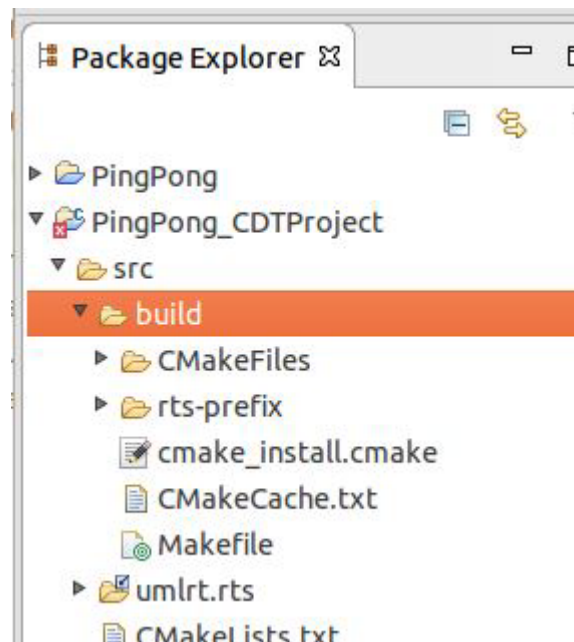
### 3. Generate the CMake structure

With the CDT project focused, run the *cmake* external tool to generate the CMake artifacts. The CMake results can be viewed in the Console window:



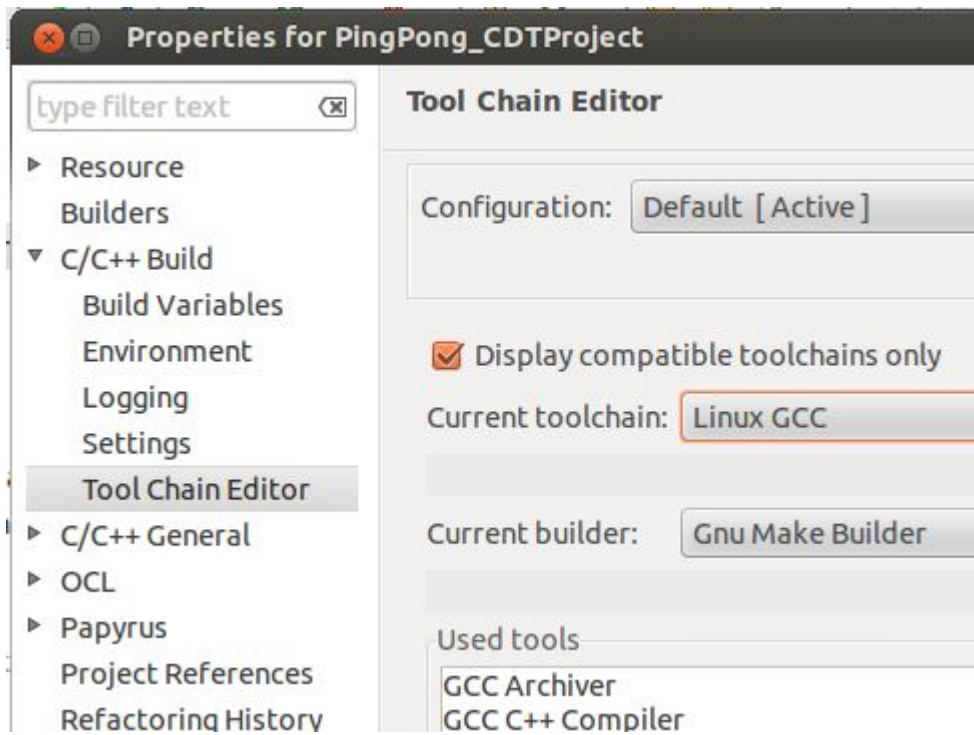
```
<terminated> cmake [Program] /usr/bin/cmake
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- RTS root: /home/william/papyrus-rt-master/runtime-PapyrusRT2/Pir
-- Detected arch, compiler, version: x86-g++-4.6.3
-- Looking for sys/socket.h
-- Looking for sys/socket.h - found
-- Looking for include files CMAKE_HAVE_PTHREAD_H
-- Looking for include files CMAKE_HAVE_PTHREAD_H - found
-- Looking for pthread_create in pthreads
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
-- Looking for pthread_mutex_timedlock in pthread
-- Looking for pthread_mutex_timedlock in pthread - found
-- home config file
-- Configuring done
-- Generating done
-- Build files have been written to: /home/william/papyrus-rt-maste
```

Refresh the build folder to see the artifacts:



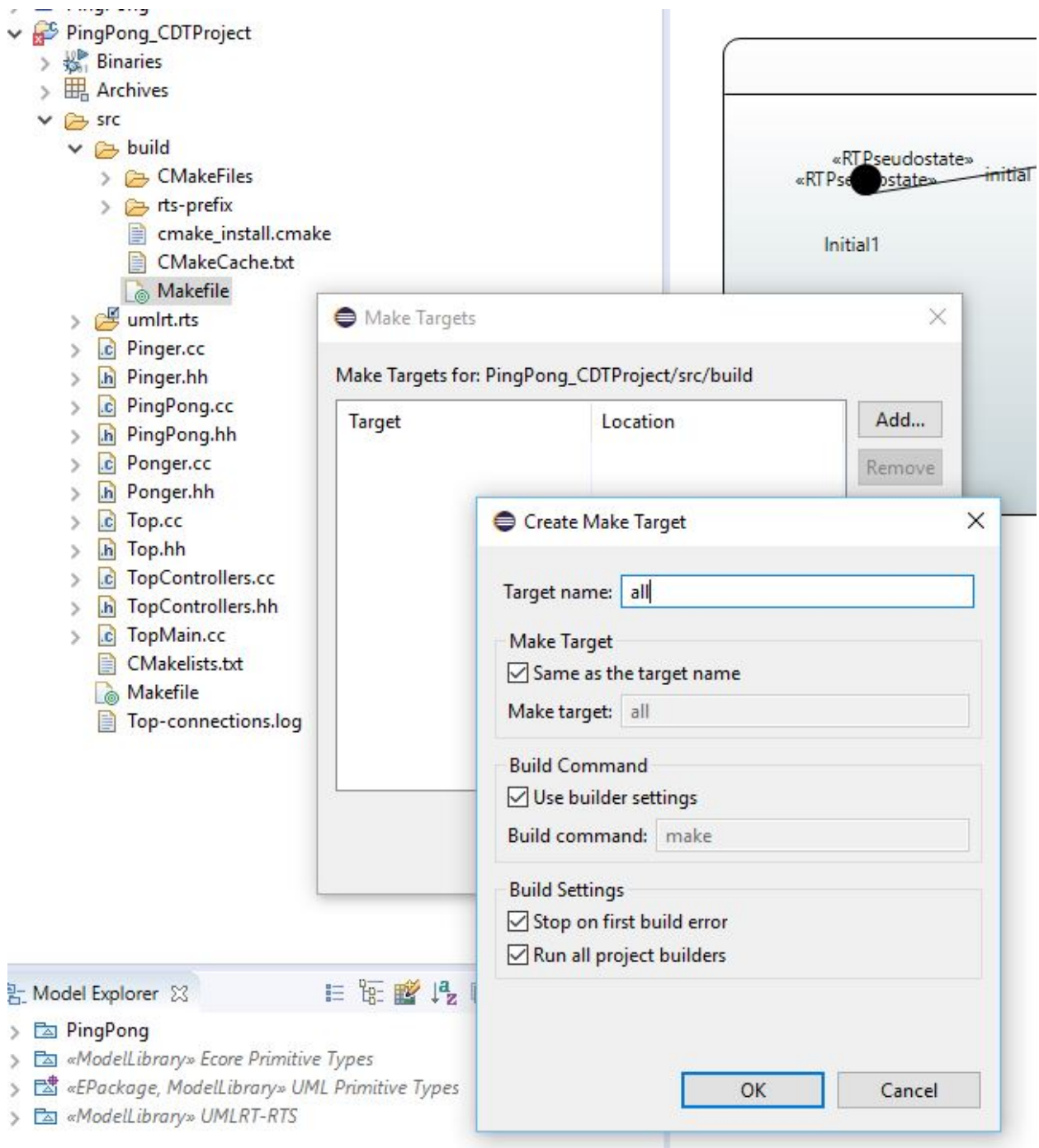
#### 4. Verify the CDT Project's Toolchain

Make sure *Linux GCC* is the project's active toolchain.



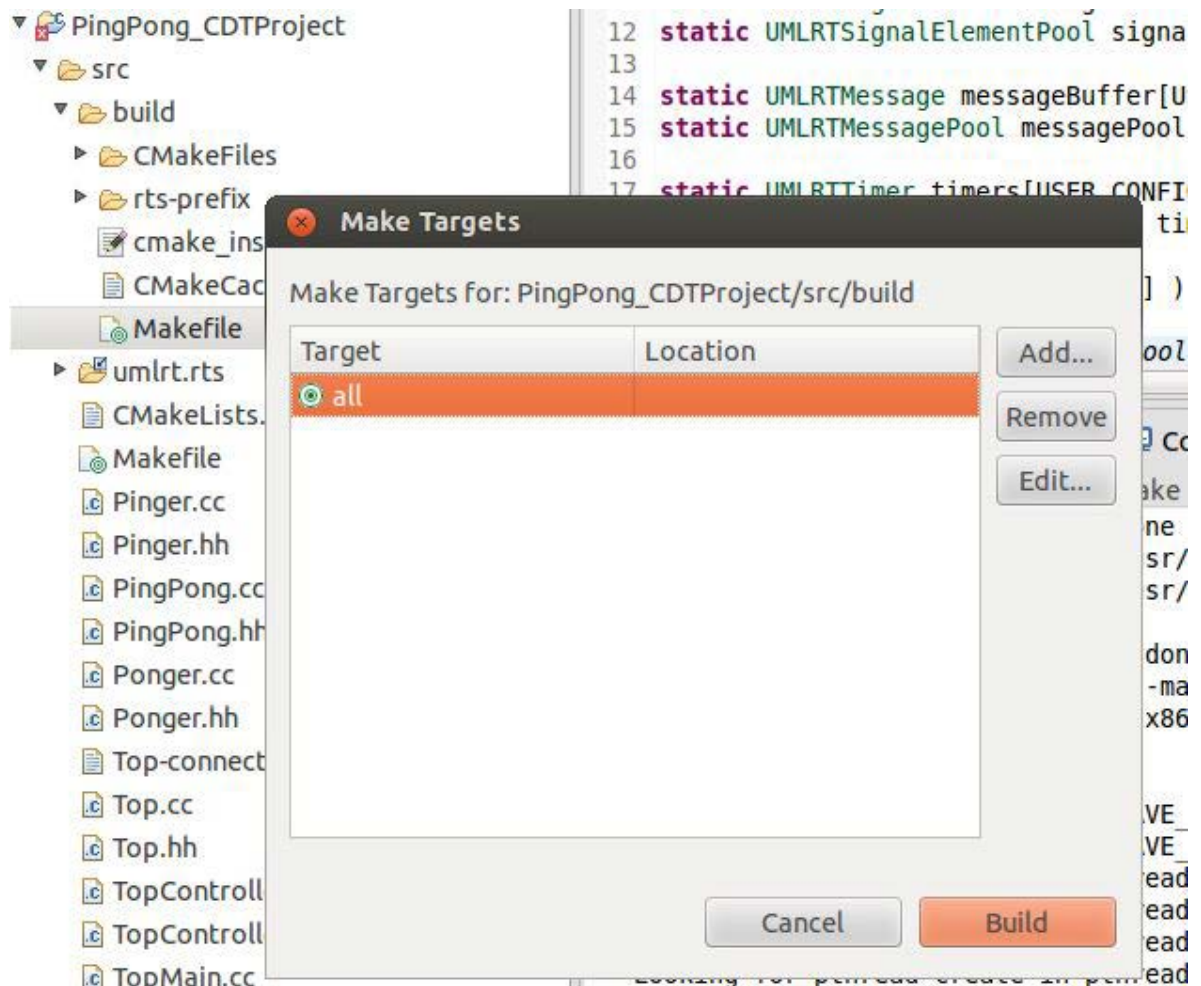
## 5. Update the Make Target

Select the Makefile generated by CMake and create the Make Target, *all*. The dialog is available via the context menu, Make Targets / Build...



## 6. Build the CDT Project

Select the target, *all*, and run the build.

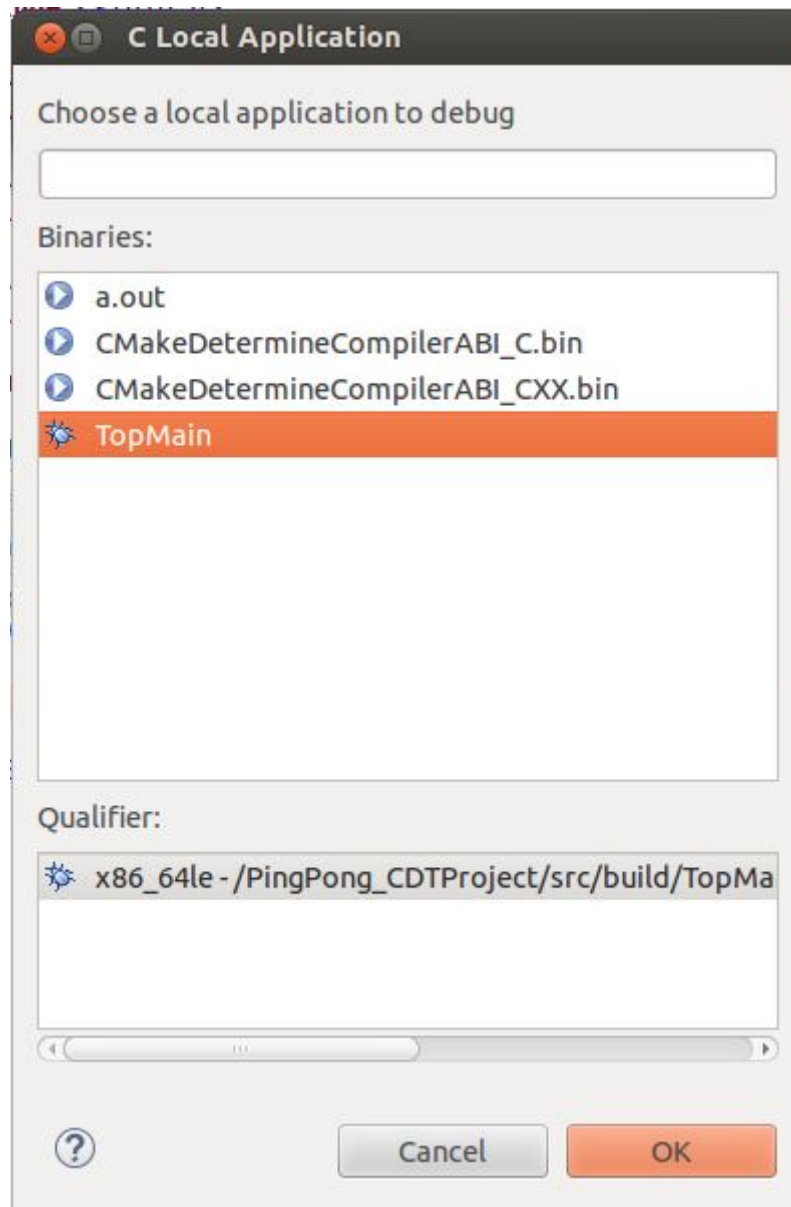


The build results can be viewed in the Console window:

```
[ 97%] Building CXX object CMakeFiles/rts.dir/os/linux/ostime.cc.o
[100%] Building CXX object CMakeFiles/rts.dir/os/linux/ostimespec.cc.o
Linking CXX static library librtsd.a
[100%] Built target rts
Install the project...
-- Install configuration: "Debug"
-- Installing: /home/william/papyrus-rt-master/runtime-PapyrusRT2/PingPong_CDTProject/
[ 57%] Completed 'rts'
[ 57%] Built target rts
Scanning dependencies of target TopMain
[ 64%] Building CXX object CMakeFiles/TopMain.dir/TopMain.cc.o
[ 71%] Building CXX object CMakeFiles/TopMain.dir/PingPong.cc.o
[ 78%] Building CXX object CMakeFiles/TopMain.dir/Pinger.cc.o
[ 85%] Building CXX object CMakeFiles/TopMain.dir/Ponger.cc.o
[ 92%] Building CXX object CMakeFiles/TopMain.dir/Top.cc.o
[100%] Building CXX object CMakeFiles/TopMain.dir/TopControllers.cc.o
Linking CXX executable TopMain
[100%] Built target TopMain
20:33:19 Build Finished (took 13s.268ms)
```

## 7. Setup a Debug Configuration

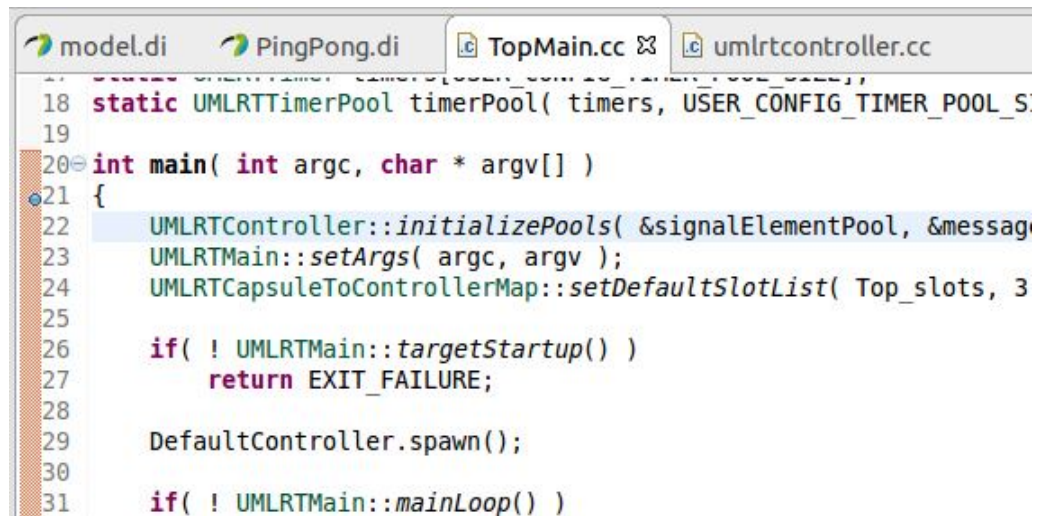
Create a C/C++ Application configuration. Select TopMain.exe if prompted.





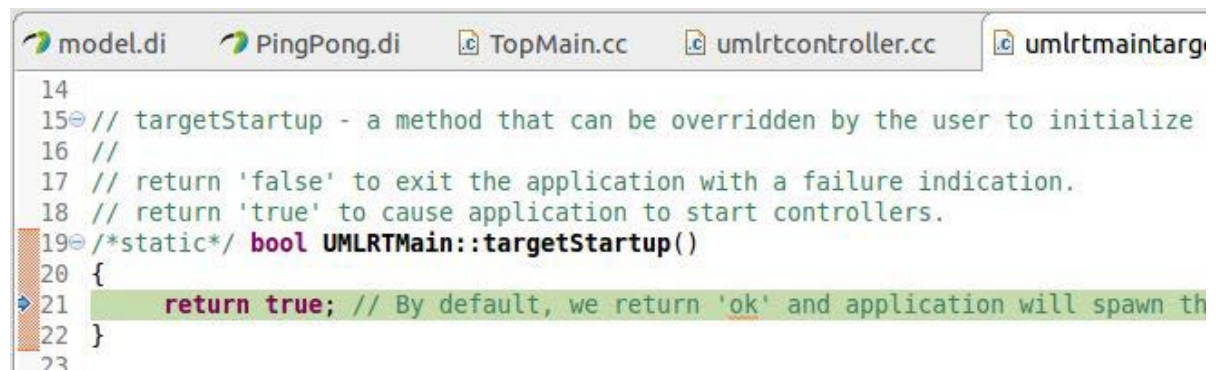
## 8. Debug the Modeled Application

Set a breakpoint at the beginning of main, and start debugging.



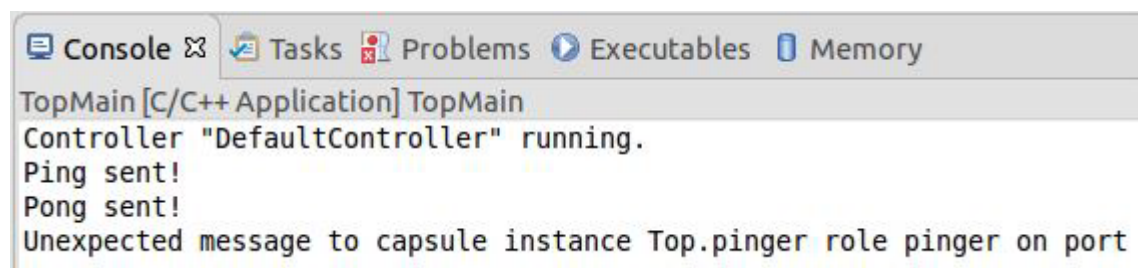
```
model.di PingPong.di TopMain.cc umlrtcontroller.cc
17 static UMLRTTimerPool timerPool( timers, USER_CONFIG_TIMER_POOL_SIZE );
18 static UMLRTTimerPool timerPool( timers, USER_CONFIG_TIMER_POOL_SIZE );
19
20 int main( int argc, char * argv[] )
21 {
22     UMLRTController::initializePools( &signalElementPool, &message
23     UMLRTMain::setArgs( argc, argv );
24     UMLRTCapsuleToControllerMap::setDefaultSlotList( Top_slots, 3
25
26     if( ! UMLRTMain::targetStartup() )
27         return EXIT_FAILURE;
28
29     DefaultController.spawn();
30
31     if( ! UMLRTMain::mainLoop() )
```

The RTS library can be stepped into as shown:



```
model.di PingPong.di TopMain.cc umlrtcontroller.cc umlrtmaintarg
14
15 // targetStartup - a method that can be overridden by the user to initialize
16 //
17 // return 'false' to exit the application with a failure indication.
18 // return 'true' to cause application to start controllers.
19 /*static*/ bool UMLRTMain::targetStartup()
20 {
21     return true; // By default, we return 'ok' and application will spawn th
22 }
23
```

The results:



```
Console Tasks Problems Executables Memory
TopMain [C/C++ Application] TopMain
Controller "DefaultController" running.
Ping sent!
Pong sent!
Unexpected message to capsule instance Top.pinger role pinger on port
```