AT&T Developer Program

Application Resource Optimization

Application Resource Optimizer (ARO) Compilation and Build Guide

Revision Date **April 18, 2012**

Legal Disclaimer

Table of Contents

Introduction

The *AT&T Application Resource Optimizer (ARO)* is a diagnostic tool for analyzing mobile web application performance. AT&T ARO allows you to automatically profile an application to optimize performance, make battery utilization more efficient, and reduce network impact.

The *ARO Data Collector* is the component of the ARO application that captures the data traffic of a mobile device and stores that information in trace files that can be analyzed using the *ARO Data Analyzer*.

The *ARO Data Analyzer* is the component of the ARO application that evaluates the trace data collected from an application and generates statistical and analytical results based on recommended best practices.

The open source code for these ARO components is available in the ARO Open Source Code Package on GitHub at https://github.com/attdevsupport/ARO. Each component can be compiled into a separate application.

The ARO Compilation and Build Guide is for developers that want to modify the open source code, or the code of any of the open source library dependencies, before compiling and building the ARO components.

The ARO Compilation and Build Guide contains the following sections:

Section	**Description**
ARO Data Collector Compilation and Build Guide	How to compile and build the open source ARO Data Collector code into an Android application package file (APK), using Apache ANT.
ARO Data Analyzer Compilation and Build Guide	How to compile and build the open source ARO Data Analyzer code into a standalone application using Apache ANT. Includes information on how to compile and build the Jpcap library for Windows and the Mac OS.
ARO Open Source Library Dependencies	Describes each of the open source libraries that are used by the ARO components.

ARO Data Collector Compilation and Build Guide

The ARO Data Collector Compilation and Build Guide describes how to compile and build the open source ARO Data Collector code into an Android application package file (APK), using Apache ANT. The APK file can then be installed on a device via the Android Debug Bridge (ADB).

Note: In order for the ARO Data Collector to be fully functional, it must be installed on a *rooted* device.

Setting Up the Development Environment

To setup your development environment so that you can compile and build the open source ARO Data Collector using Apache Ant, follow these steps:

Step 1: Download and setup the Android SDK. Go to the Android Developers website http://developer.android.com/sdk/index.html, download the Android package for your platform, and follow the steps to setup Android SDK.

Step 2: Add additional components. Follow the installation instructions at http://developer.android.com/sdk/installing.html, especially the step: *Adding Platforms and Other Components.*

Step 3: Download Apache ANT. Go to http://ant.apache.org and follow the download instructions.

Compiling and Building ARO Data Collector

To compile and build the open source ARO Data Collector code using Android Ant, Use the following steps:

Step 1: Checkout the ARO Data Collector source code to a local folder.

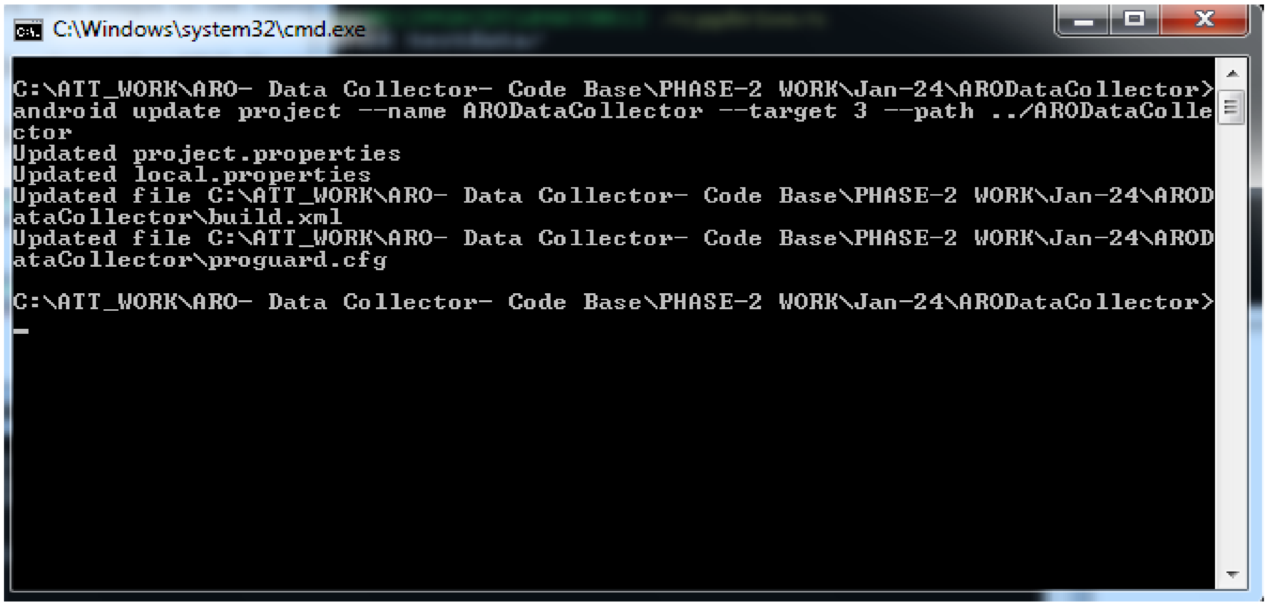
Step 2: Update the project.properties to match your local configuration and generate a **build.xml**. This can be done from a command prompt using the following syntax.

```
android update project --name <project_name> --target <target_ID>
--path <path_to_your_project>
```

Note: ARO Data Collector supports Android 2.1 and above, so the target id should be set to 2.1 and above. In the following example it is set to 7.

```
android update project --name granted
```

The following example shows the project.properties being updated and the **build.xml** being generated:



C:\Windows\system32\cmd.exe

```
C:\ATT_WORK\ARO- Data Collector- Code Base\PHASE-2 WORK\Jan-24\ARODataCollector>
android update project --name ARODataCollector --target 3 --path ../ARODataColle
ctor
Updated project.properties
Updated local.properties
Updated file C:\ATT_WORK\ARO- Data Collector- Code Base\PHASE-2 WORK\Jan-24\AROD
ataCollector\build.xml
Updated file C:\ATT_WORK\ARO- Data Collector- Code Base\PHASE-2 WORK\Jan-24\AROD
ataCollector\proguard.cfg

C:\ATT_WORK\ARO- Data Collector- Code Base\PHASE-2 WORK\Jan-24\ARODataCollector>
```

Step 3: The ARO Data Collector project can be built for debug or release using the *ant* command.

To build a debug version of the ARO Data Collector use the command:

```
ant debug
```

To build a release version of the ARO Data Collector use the command:

```
ant release
```

Step 4: Transfer the ARO Data Collector APK file to a *rooted* device using the following command:

```
adb aro
```

ARO Data Analyzer Compilation and Build Guide

The ARO Data Analyzer Compilation and Build Guide describes how to compile and build the open source ARO Data Analyzer code using Apache ANT to generate a standalone Java application. It describes how to set up the required development environment, check out and compile the open source code, and build the application.

Note: A build of the ARO Data Analyzer application (**aro.jar**) is included in the ARO Open Source Code Package in the **ARODataAnalyzer\lib** directory.

Setting Up the Development Environment

To setup your development environment for compiling and building the open source ARO Data Analyzer, ensure that you meet the following system requirements for your OS:

The system requirements for compiling, building, and running the ARO Data Analyzer on a Windows OS are:

- A computer running Windows XP, Windows Vista, or Windows Seven.
- At least 1GB of RAM.
- The Java Runtime Environment (JRE) version 1.6 or greater is required to run the ARO Data Analyzer. To compile and build the ARO Data Analyzer in the same environment, install the Java Development Kit (JDK) version 1.6 or greater which includes the JRE.
- WinPcap, the “industry-standard windows packet capture library”.
- ANT, An Apache Java build tool.

Note: If needed, configure the JAVA_HOME system variable so that it points to your Java installation directory. This can be done in the Advanced tab of System Properties, by editing the Environment Variables.

The system requirements for compiling, building, and running the ARO Data Analyzer on the Mac OS are:

- A computer running Mac OS X 10.6 and above.
- At least 1GB of RAM.
- The Java Runtime Environment (JRE) version 1.6 or greater is required to run the ARO Data Analyzer. To compile and build the ARO Data Analyzer in the

same environment, install the Java Development Kit (JDK) version 1.6 or greater which includes the JRE.

- ANT, An Apache Java build tool.

Compiling and Building ARO Data Analyzer For Java

To compile and build the open source ARO Data Analyzer code for Java, do the following:

Checkout the ARO Data Analyzer source code including the **build.xml** file, from the **ARODataAnalyzer** folder to a local folder, and use ANT to compile and build the application.

The **build.xml** file builds the **.jar** file that contains the compiled classes for the ARO Data Analyzer.

The files **aro.bat** and **aro** are included in the **ARODataAnalyzer\lib** directory of the ARO Open Source Package to help you run the **.jar** file with default run settings:

- To launch the **.jar** file in the Windows OS, run **aro.bat** from the command line.
- To launch the **.jar** file in the Mac OS, run **aro**.

Compiling and Building the Jpcap Library

The ***Jpcap*** library is a custom JNI library that forms a wrapper around the functionality of the ***tcpdump*** and ***libpcap*** libraries so that they can be used in Java applications. The following sections describe how to compile and build a Jpcap library for use with the Windows and the Mac operating systems.

Compiling and Building the Jpcap Library for Windows

The following section describes how to compile and build a ***Jpcap*** library for the Windows operating system. It lists the system requirements, and provides instructions for compiling and building Jpcap.dll (32-bit), or Jpcap64.dll (64-bit).

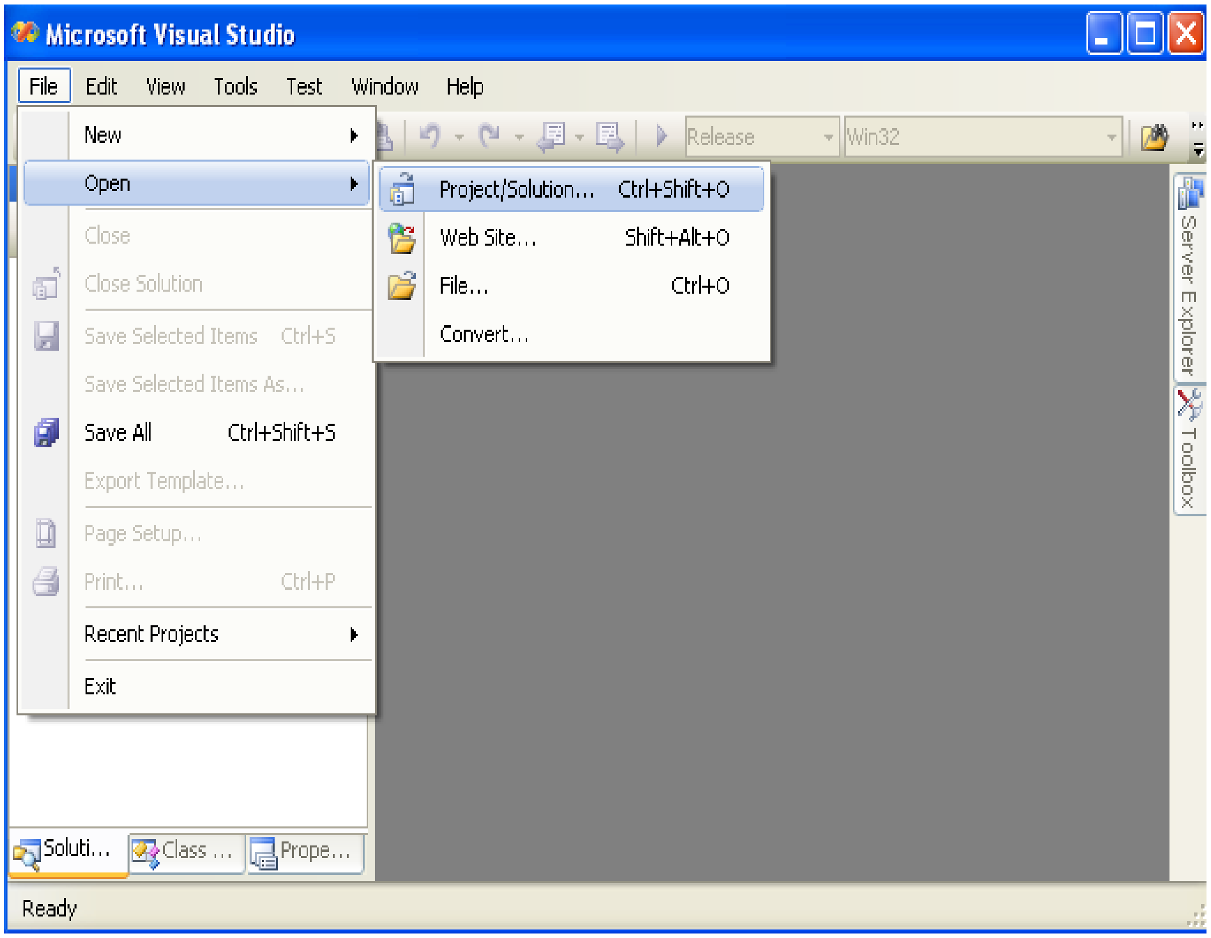
System Requirements:

- A computer running Windows XP, Windows Vista, or Windows Seven.
- At least 1GB of RAM.
- Microsoft Visual Studio 2008/2010
- Java Development Kit (JDK 1.6/1.7)

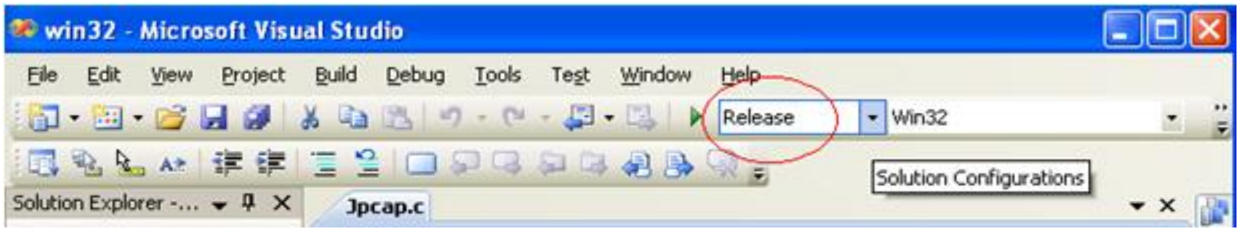
To compile and build a Windows Jpcap library using Microsoft Visual Studio, do the following:

Step 1: Checkout the *Jpcap* source code from the ARODataAnalyzer\c directory of the Open Source code package, to a local folder.

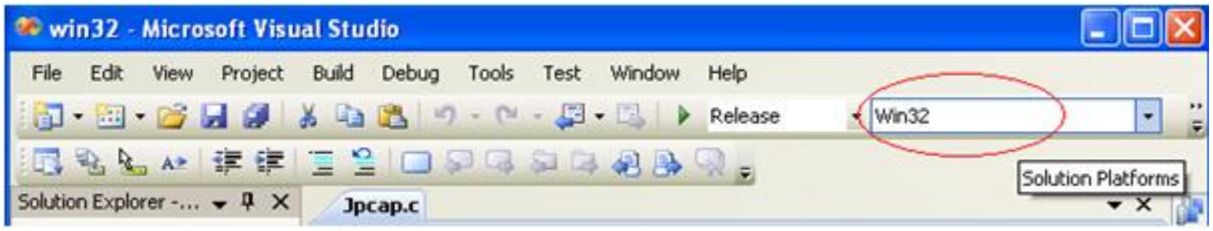
Step 2: Start Microsoft Visual Studio and open the Jpcap project solution file (\c\win32\win32.sln) by selecting File menu -> Open -> Project/Solution. The following example shows the menu selection:



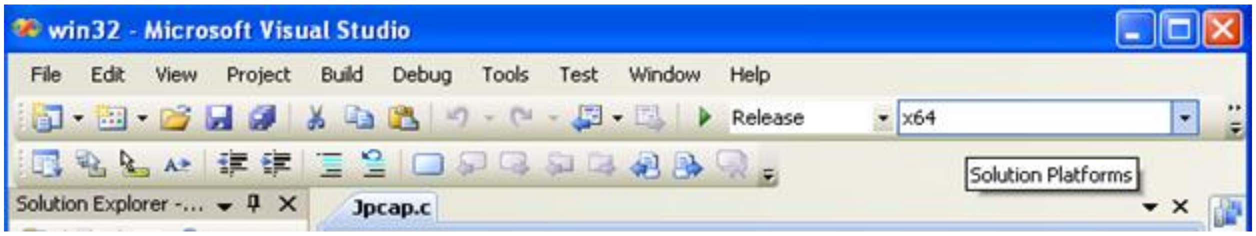
Step 3: Select “Release” in the Solution Configurations options as in the following example:


win32 - Microsoft Visual Studio
File Edit View Project Build Debug Tools Test Window Help
Release
Win32
Solution Configurations
Solution Explorer -...
Jpcap.c

Step 4: To generate a Jpcap.dll (32 bit), select "Win32" in the Solution Platforms options as in the following example:

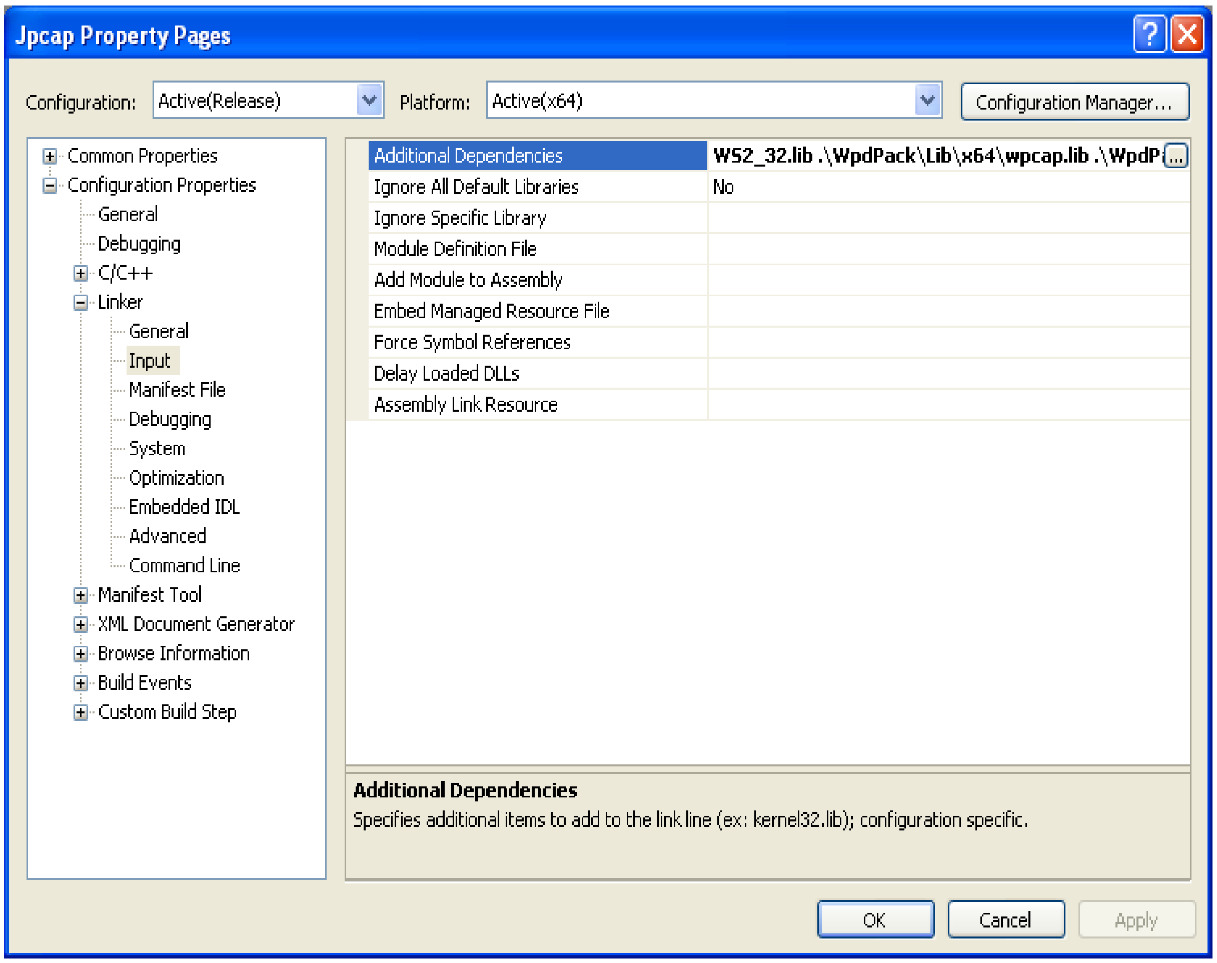

win32 - Microsoft Visual Studio
File Edit View Project Build Debug Tools Test Window Help
Release
Win32
Solution Platforms
Solution Explorer -...
Jpcap.c

Step 5: To generate a Jpcap64.dll (64-bit), select "x64" in the Solution Platforms options as in the following example.


win32 - Microsoft Visual Studio
File Edit View Project Build Debug Tools Test Window Help
Release
x64
Solution Platforms
Solution Explorer -...
Jpcap.c

Step 6: Go to Project Menu -> Jpcap Properties -> Configuration properties -> Linker -> Input -> Additional Dependencies and replace:
wpcap.lib with ***.\WpdPack\Lib\x64\wpcap.lib*** and
Packet.lib with ***.\WpdPack\Lib\x64\Packet.lib***

The following example shows this edit to the Additional Dependencies field of the Jpcap Property pages:

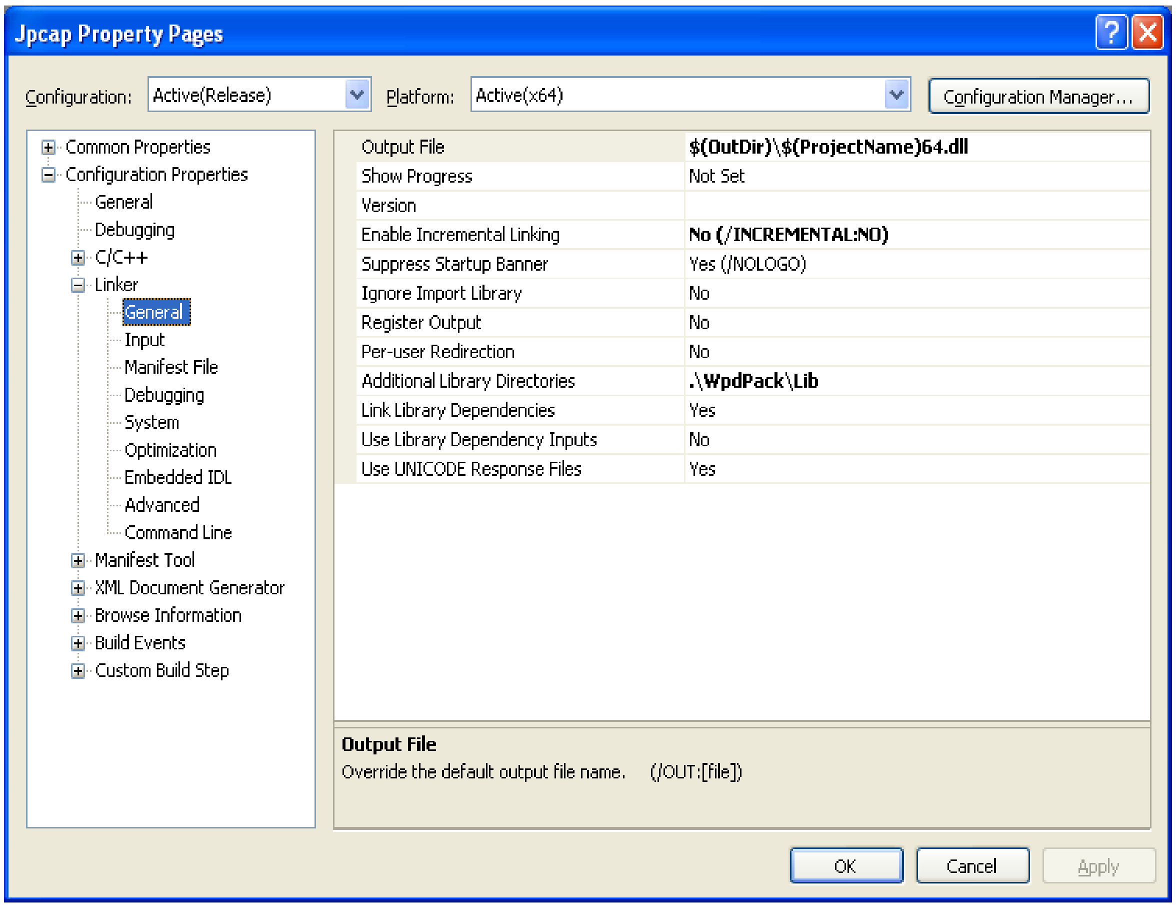


Additional Dependencies	WS2_32.lib .\WpdPack\Lib\x64\wpcap.lib .\WpdP
Ignore All Default Libraries	No
Ignore Specific Library	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.

Step 7: Go to Project Menu -> Jpcap Properties -> Configuration properties -> Linker –> General -> Output File and replace:
$(OutDir)\$(ProjectName).dll with ***$(OutDir)\$(ProjectName)64.dll***

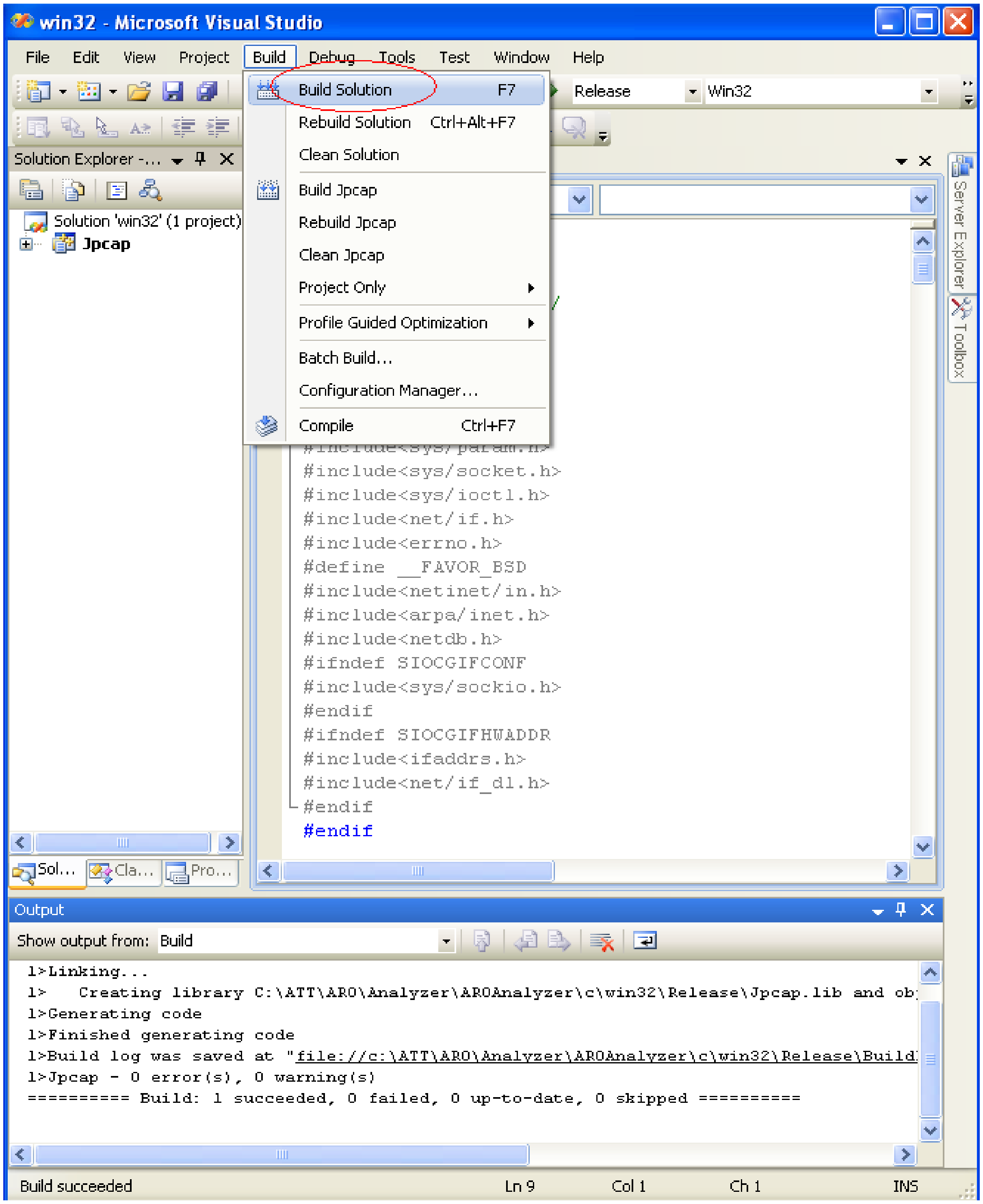
The following example shows the Output File name property being changed in the Jpcap Property pages:



Output File	**$(OutDir)\$(ProjectName)64.dll**
Show Progress	Not Set
Version	
Enable Incremental Linking	**No (/INCREMENTAL:NO)**
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	**.\WpdPack\Lib**
Link Library Dependencies	Yes
Use Library Dependency Inputs	No
Use UNICODE Response Files	Yes

Output File
Override the default output file name. (/OUT:[file])

Step 8: Select “Build Solution” from the Build options to begin building the project. The following example shows this menu selection:



```
#include<sys/param.h>
#include<sys/socket.h>
#include<sys/ioctl.h>
#include<net/if.h>
#include<errno.h>
#define __FAVOR_BSD
#include<netinet/in.h>
#include<arpa/inet.h>
#include<netdb.h>
#ifndef SIOCGIFCONF
#include<sys/sockio.h>
#endif
#ifndef SIOCGIFHWADDR
#include<ifaddrs.h>
#include<net/if_dl.h>
#endif
#endif
```

```
1>Linking...
1>   Creating library C:\ATT\ARO\Analyzer\AROAnalyzer\c\win32\Release\Jpcap.lib and obj
1>Generating code
1>Finished generating code
1>Build log was saved at "file://c:\ATT\ARO\Analyzer\AROAnalyzer\c\win32\Release\Build
1>Jpcap - 0 error(s), 0 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

If you are on a 32-bit system, Jpcap.dll is built and placed in the “Release” folder. If you are on a 64-bit system, Jpcap64.dll is built and placed in the “x64\Release” folder.

Dependencies:
The following libraries and header files are used internally when building Jpcap.dll and Jpcap64.dll:

WinPcap Developer’s Pack
JNI (Java Native Interface)

Compiling and Building a Jpcap Library for the Mac

The following section describes how to compile and build a ***Jpcap*** library for the Mac operating system. It lists the system requirements, and provides instructions for compiling and building *libjpcap.jnilib*.

System Requirements:

- A computer running Mac OS X 10.6 and above.
- At least 1GB of RAM.
- Java Development Kit (JDK 1.6/1.7)

To compile and build a Jpcap library for the Mac, do the following:

Step 1: Checkout the Jpcap source code from the ARODataAnalyzer\c directory of the Open Source code package, to a local folder.

Step 2: Make sure that Makefile is present in the same folder.

Step 3: Open command terminal and go to the source code folder path.

Step 4: Execute the “Make” command in command terminal.

Step 5: ***libjpcap.jnilib*** is built and placed in the current folder.

Dependencies:
Ensure that the Java Development Kit (JDK 1.6/1.7) is installed on your system, and that JAVA_HOME is set in the **bash_profile** file.

ARO Open Source Library Dependencies

The open source code for the ARO Data Collector and ARO Data Analyzer makes use of other open source libraries to provide certain functionality.

The ARO Data Collector uses:

- tcpdump
- libpcap
- FFmpeg

The ARO Data Analyzer uses:

- ddmlib
- JFreeChart
- FFmpeg

Notice that FFmpeg is used by both components. The FFmpeg section describes how it is configured differently for each component.

The tcpdump and libpcap libraries are both modified with new files added to them in order to be used with the ARO Data Collector. These modifications are explained in detail in the tcpdump and libpcap sections.

The following sections describe each of these libraries and what they are used for in the ARO components.

ddmlib

The code library ***ddmlib*** contains functions which allow an application to form a connection to the host side Android Debug Bridge (ADB). The ADB provides a central point to communicate with any connected Android devices, emulators, or the applications running on those devices and emulators.

Use in the ARO Data Analyzer:
ddmlib is used to establish a bridge between the ARO Data Analyzer and the Android emulator to manage the collection of trace data from the emulator.

Available at:

ddmlib is part of the Android SDK tools library, and can be extracted from the directory: <*Android SDK Path*>\tools\lib. For more information about ***ddmlib***, or to download it, go to the Android project tools link: http://tools.android.com/overview/sdk

License:
The license for ddmlib is available at: http://source.android.com/source/licenses.html.

JFreeChart

JFreeChart is a free Java chart library that can be used to display charts in an application. It is a well supported library with fully documented APIs, providing a wide range of chart types that are easy to extend.

Use in the ARO Data Analyzer:
The ***JfreeChart*** library is used in the ARO Data Analyzer to display the Diagnostics View chart for loaded trace files.

Available at:
The ***JfreeChart*** library is available at: http://www.jfree.org/jfreechart/download.html

License:
http://www.gnu.org/licenses/lgpl.html

FFmpeg

FFmpeg is a complete, cross-platform solution to record, convert, and stream audio and video. ***FFmpeg*** provides various tools, including:

- A command line tool for converting multimedia files between formats.
- A command line tool for capturing and recording video of a screen display.

The FFmpeg library is used in both ARO components.

Use in the ARO Data Analyzer:
The ***FFmpeg*** library is used to convert the trace video files from the *.mp4* format to the *.mov* format.

Use in the ARO Data Collector:
The ***FFmpeg*** library is used to record the device screen video by directing the input from the screen buffer to a file.

Available at
FFmpeg is available as a static build, or as open source code that is downloaded and configured using compilation parameters. The ARO components make use of both methods.

The ARO Data Analyzer uses a 32-bit static build of ***FFmpeg*** for Windows available at:
http://ffmpeg.zeranoe.com/builds/

The ARO Data Collector uses ***FFmpeg*** source code version 0.8 from the following location:

svn checkout svn://svn.ffmpeg.org/ffmpeg/trunk ffmpeg

The source code from this location is compiled on linux with the following configuration parameters:

```
$ ./configure --target-os=linux --enable-cross-compile --enable-static --
disable-shared --disable-asm --disable-yasm --disable-ffprobe --disable-
ffserver --disable-ffplay --disable-indev=v4l2 --disable-indev=v4l --disable-
indev=oss --disable-indev=dv1394 --cc=arm-none-linux-gnueabi-gcc --disable-
outdevs --disable-protocols --disable-bsfs --disable-network --disable-muxers
--disable-demuxers --disable-parsers --disable-decoders --disable-encoders --
enable-protocol=file --enable-muxer=mp4 --enable-parser=mpeg4video --enable-
decoder=mpeg4 --enable-decoder=rawvideo --enable-encoder=mpeg4 --arch=arm
```

Note: ***FFmpeg*** is compiled without using the flags *–enable-gpl* and *–enable-nonfree*.

License:
http://ffmpeg.org/legal.html

tcpdump & libpcap

The ***tcpdump*** library contains functionality that provides a powerful command-line packet analyzer.

libpcap is a portable C/C++ library that provides functionality for network traffic capture.

This section describes how to build these two libraries for use in an Android application. To use tcpdump or libpcap with a Windows or Mac application, see the section *Compiling and Building a Jpcap Library*.

Use in the ARO Data Analyzer:

The ***tcpdump*** binary is used in the ARO Data Collector to capture the network traffic (tcp/udp) packets on the device.

Versions:
The ARO Data Collector uses the following library versions of ***tcpdump*** and ***libpcap***:

- libpcap 1.1.1
- tcpdump 4.1.1

Additions and Modifications to the source code:
The open source code of the ARO Data Collector contains the following additions and modifications to the ***tcpdump*** and ***libpcap*** libraries.

The following files have been added to the ***tcpdump*** source code:

- **socket_server.h**: The server socket header file.
- **socket_server.c**: Provides a server socket that starts and stops the ***tcpdump***.
- **Android**.**mk**: The Android NDK make file.

The following file is modified in the ***tcpdump*** source code:

- **tcpdump.c**: The source code is modified to ignore tcp/udp packets after the EXIT command is received on the socket for *pcaploop*.

The following files have been added to the ***libpcap*** source code:

- **user_input.h**: The input header file.
- **user_input.c**: A source file containing functions that capture the user events on a device during a pcap loop. The information is read into ***events*** files, and is checked against a key mapped database.
- **socket_proc.h**: The socket_proc header file.
- **socket_proc.h**: A source file containing functions that capture the application name during a pcap loop along with a time trace file. Capturing these items together, records a synchronized time stamp of traffic.cap.
- **Android**.**mk**: The Android NDK make file.

The following file is modified inside the ***libpcap*** source code:

- **pcap-linux.c:** The source code is modified to track the user event mappings along with the app id, and appname trace collection, during a pcap loop.

Note: The code changes have been marked by ***CODE CHANGE*** inside the source file.

Code library compilation prerequisites:

The following software is required in order to compile tcpdump and libpcap with the previously mentioned additions and modifications:

- The Android NDK
- Cygwin: A tool for compiling code using the Android NDK.

Code library compilation steps:
The following describes how to create a make file for tcpdump and libpcap, and how to compile a new tcpdump binary.

Creating the Make File:
An *Android.mk* or "make file" describes your code source files to the build system. Detailed information on how to create an Android.mk file can be found at android-ndk-xxx\docs\ANDROID-MK.html.

Modifying an existing Make File:
To modify the existing Android.mk to add new files for libcap and tcpdump, you need to add the new file names to the list of LOCAL_SRC_FILES variables. This is the list of source files that will be built.

The following example shows how a new file *foo.c* is added to the LOCAL_SRC_FILES variables.

The **LOCAL_SRC_FILES** variables contain a list of C files that will be built.

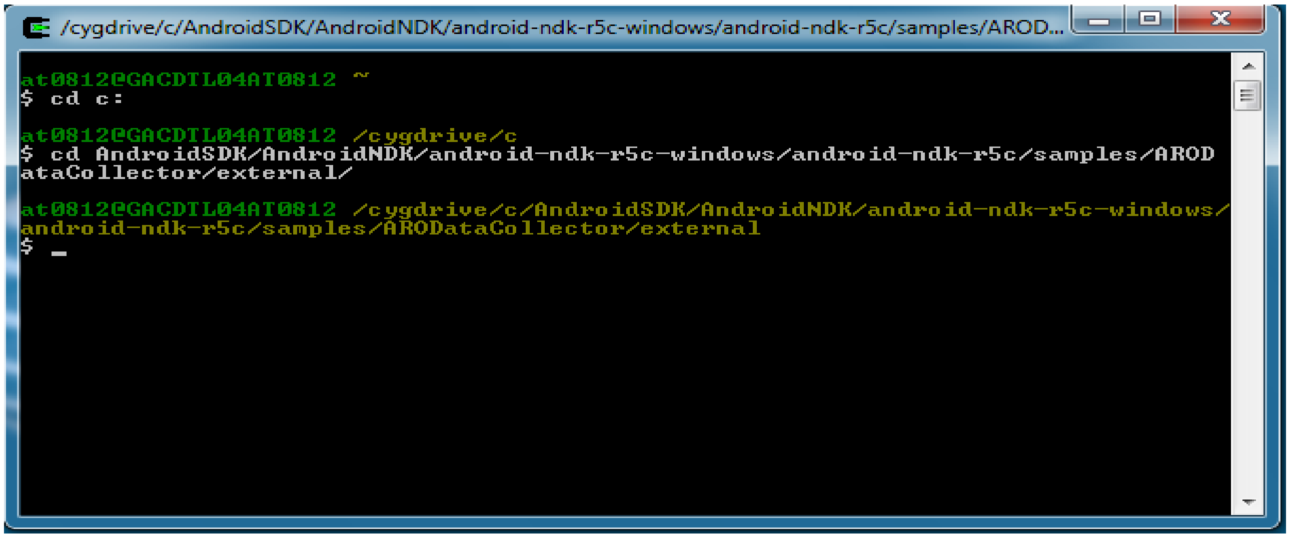
```
LOCAL_SRC_FILES:=\
      bpf_image.c\
      etherent.c\
      fad-gifc.c\
      foo.c\
      gencode.c\
```

Compiling tcpdump:
The following steps describe how to compile the ***tcpdump*** binary:

Step 1: Set the environment variable **%ANDROID_NDK_ROOT%** so that it points to the root directory of Android NDK.

Step 2: Checkout the ARO Data Collector project from GitHub.

Step 3: Open the **Cygwin** command prompt and navigate to the **external** folder inside the ARO Data Collector project that contains the tcpdump/libpcap source code files. The following example shows this step.


/cygdrive/c/AndroidSDK/AndroidNDK/android-ndk-r5c-windows/android-ndk-r5c/samples/AROD...

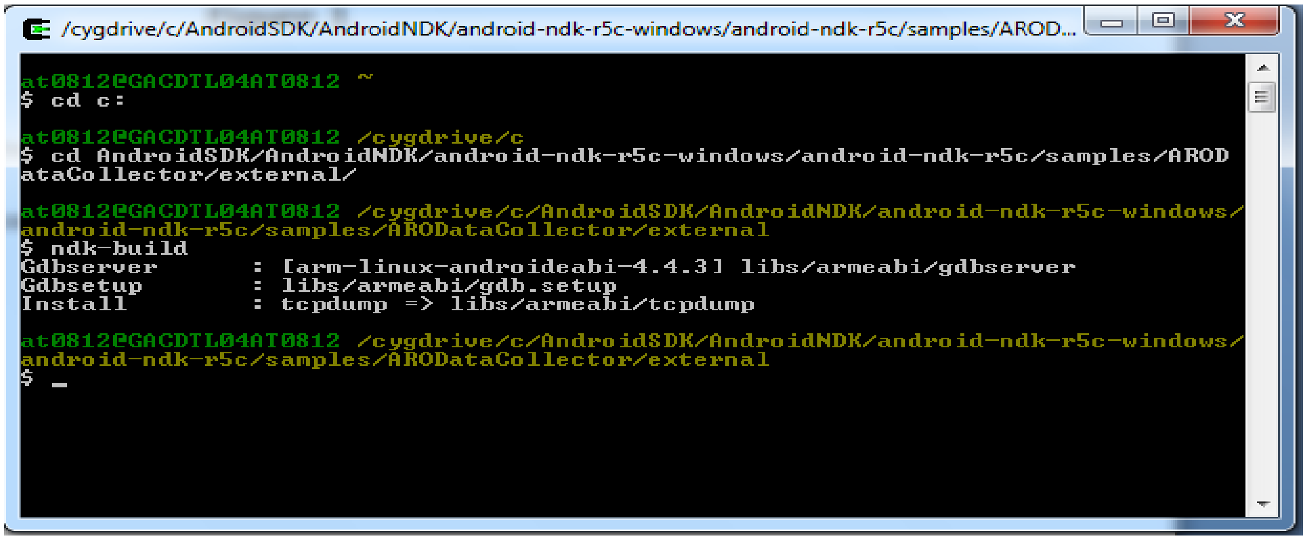
```
at0812@GACDTL04AT0812 ~
$ cd c:

at0812@GACDTL04AT0812 /cygdrive/c
$ cd AndroidSDK/AndroidNDK/android-ndk-r5c-windows/android-ndk-r5c/samples/AROD
ataCollector/external/

at0812@GACDTL04AT0812 /cygdrive/c/AndroidSDK/AndroidNDK/android-ndk-r5c-windows/
android-ndk-r5c/samples/ARODataCollector/external
$ _
```

Step 4: Create the make file *ARODataCollector/external/jni/Android.mk* to describe the native sources to the NDK build system.

Step 5: Build the ***tcpdump*** native code by running the **'ndk-build'** script from the *ARODataCollector/external* directory which generates a new ***tcpdump*** binary under the */libs/armeabi/* folder. The following example demonstrates this.


/cygdrive/c/AndroidSDK/AndroidNDK/android-ndk-r5c-windows/android-ndk-r5c/samples/AROD...

```
at0812@GACDTL04AT0812 ~
$ cd c:

at0812@GACDTL04AT0812 /cygdrive/c
$ cd AndroidSDK/AndroidNDK/android-ndk-r5c-windows/android-ndk-r5c/samples/AROD
ataCollector/external/

at0812@GACDTL04AT0812 /cygdrive/c/AndroidSDK/AndroidNDK/android-ndk-r5c-windows/
android-ndk-r5c/samples/ARODataCollector/external
$ ndk-build
Gdbserver      : [arm-linux-androideabi-4.4.3] libs/armeabi/gdbserver
Gdbsetup       : libs/armeabi/gdb.setup
Install        : tcpdump => libs/armeabi/tcpdump

at0812@GACDTL04AT0812 /cygdrive/c/AndroidSDK/AndroidNDK/android-ndk-r5c-windows/
android-ndk-r5c/samples/ARODataCollector/external
$ _
```

Available at:
The tcpdump and libpcap source code is available at:
http://www.tcpdump.org/#latest-release

License:
The license for tcpdump is available at: http://www.tcpdump.org/license.html.