

Getting Started with the IxChariot Proxy Endpoint

Introduction

The market for network-connected devices is experiencing tremendous growth and rapid change. To squeeze every ounce of performance out of these devices, vendors frequently use highly-specialized microcontrollers and lightweight real-time operating systems. Moving to a new platform typically means switching to an entirely different toolset—impacting time-to-market.

What is the Proxy Endpoint?

The IxChariot Proxy Endpoint is a software development kit (SDK) that enables the development of custom endpoints for IxChariot. As shown in Figure 1, the IxChariot Proxy Endpoint works by splitting Ixia’s mainstream endpoint agent into two pieces:

- Proxy Endpoint — runs on a PC and handles statistics and test management
- DUT shim — device-resident agent that performs traffic generation

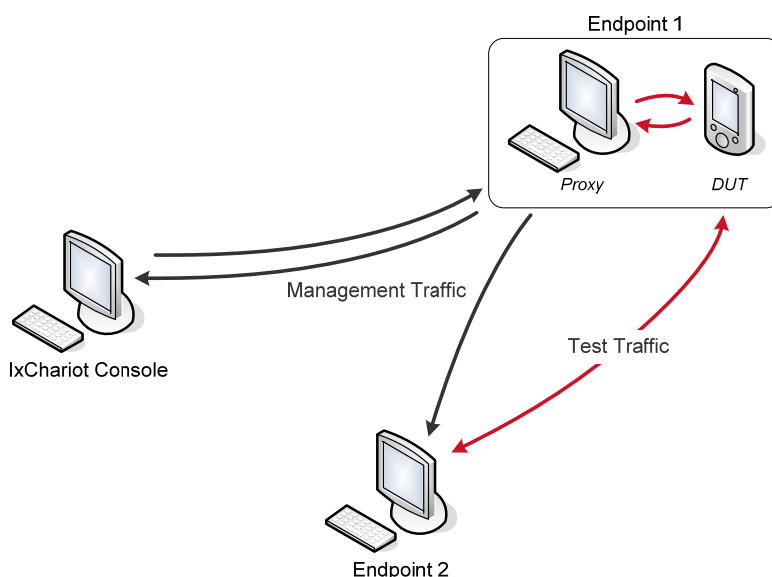


Figure 1. IxChariot Proxy Endpoint

The Proxy Endpoint SDK includes the following components:

- Full source code for a 'C'-language DUT shim device code for Linux- and Windows-based embedded platforms
- Complete proxy/device protocol specification for highly-customized applications

The IxChariot Proxy Endpoint architecture has several advantages for customers developing networked devices with custom applications:

- Compatible with IxChariot's extensive library of application, voice and video scripts
- Works with existing IxChariot Performance Endpoints
- Management overhead is off-loaded to a partner PC
- Single-threaded example code makes minimal use of OS calls—sockets and timers only
- Fixed memory allocation to accommodate platforms with limited memory
- Source code enables developers to use platform-specific operating system features to achieve maximum performance.

This document will take you step-by-step through the process of building and using the Proxy Endpoint SDK for wireless device testing. Additionally, a number of tests were run on a typical consumer device to give users an idea of the expected performance under real-world conditions.

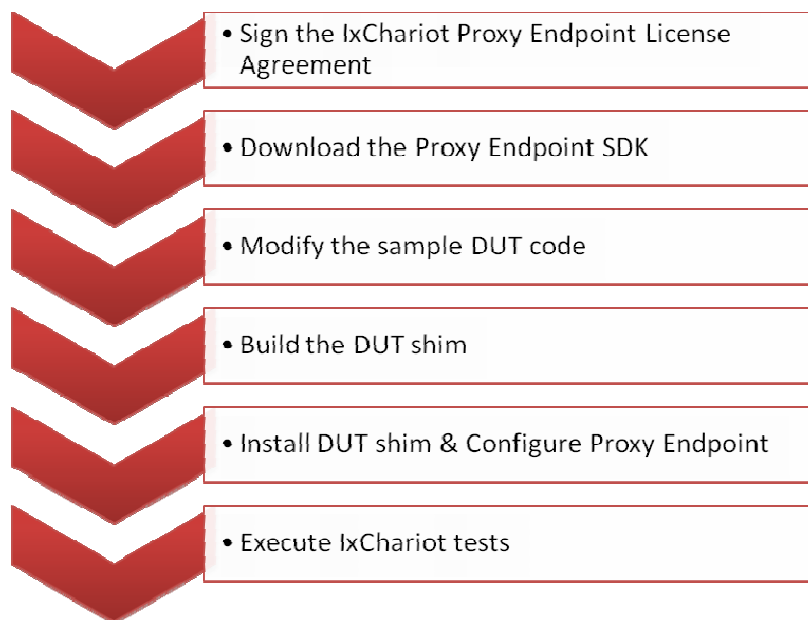


Figure 2. Proxy Endpoint Development Process.

Building the Proxy Endpoint

Step 1: Configure the target toolchain

The first step is to download the toolchain and install it on the build host system. In this case we'll use the CodeSourcery GNU/Linux toolchain for ARM EABI targets – currently a very popular platform for mobile devices.

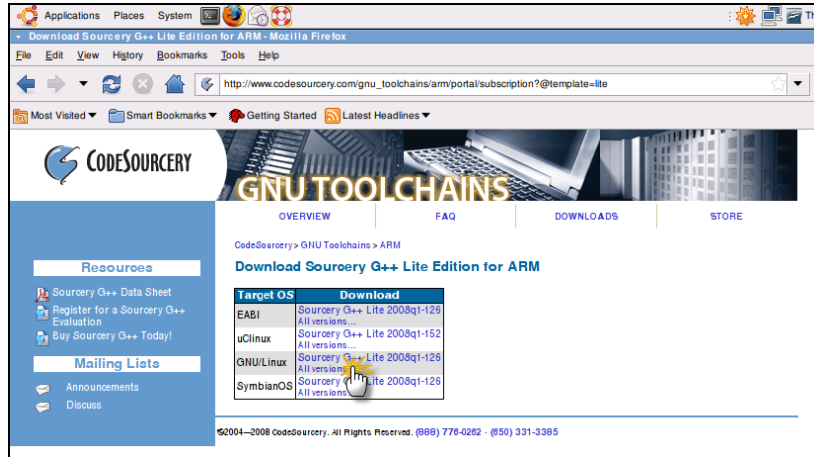


Figure 3. Downloading the toolchain.

After expanding the archive, the actual build tools can be found in 'arm-2008q1/bin'. To build the DUT shim, we will modify the makefile to point to these utilities.

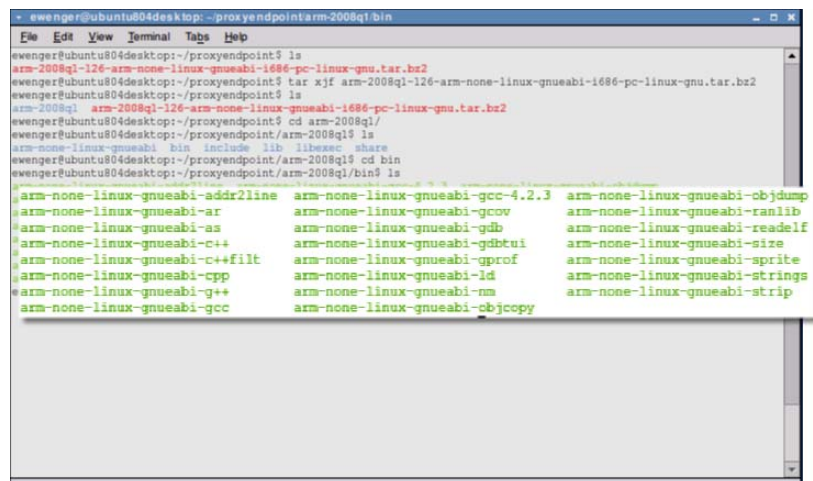


Figure 4. CodeSourcery's toolchain binaries.

Step 2: Install the IxChariot Proxy Endpoint SDK

When you return the signed IxChariot Proxy Endpoint License Agreement, you will receive the Proxy Endpoint SDK. The SDK is delivered in a compressed archive called ProxyEndpoint_SDK.tar.gz.

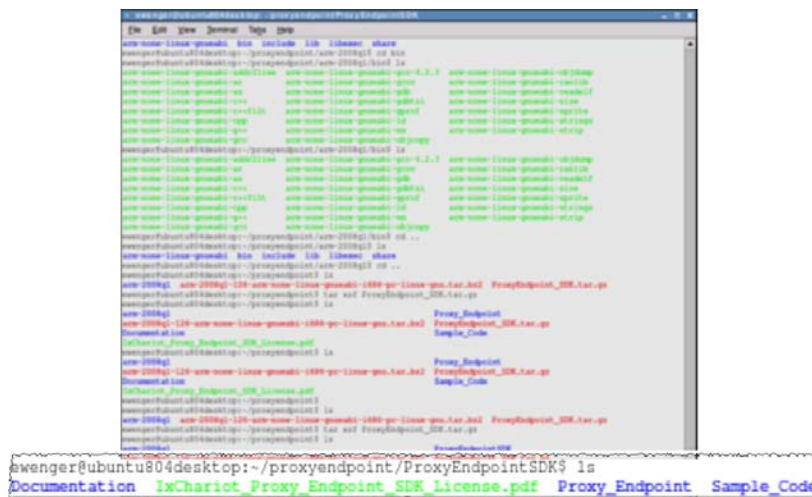


Figure 5. Unpacking the Proxy Endpoint tarball.

After uncompressing this to the desired location you will have the following directory structure on your filesystem:

Filename	Description
ProxyEndpointSDK/	
Documentation/	User Guide & Protocol Specification
Proxy_Endpoint/	Proxy Endpoint to be installed on Windows PC
Sample_Code/	Sample code for the DUT shim
IxChariot_Proxy_Endpoint_SDK_License.pdf	Copy of the license agreement

Table 1. Proxy Endpoint directory structure.

Step 3: Edit the DUT Shim Makefile

The Proxy Endpoint SDK contains a number of example makefiles for various systems. The best one to start with is generic Linux makefile, *makefile_ds.lnx.gcc*, which can easily be modified to work with a cross-compiler. The first step is to create a copy of the generic makefile to use as a scratchpad for the new DUT shim project.

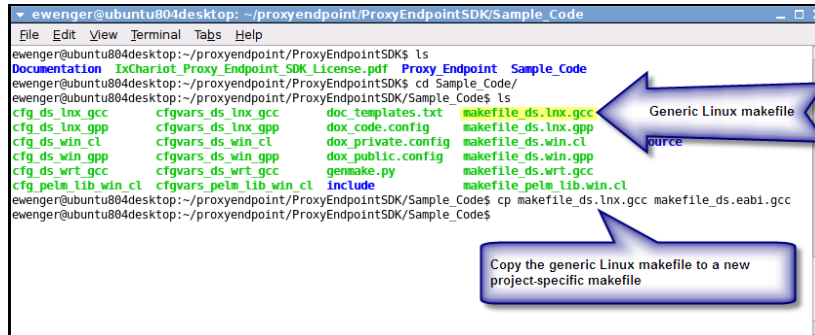


Figure 6. Create a copy of the generic Linux makefile.

Next, modify the new makefile to use the new toolchain as shown in Figure 7 below.

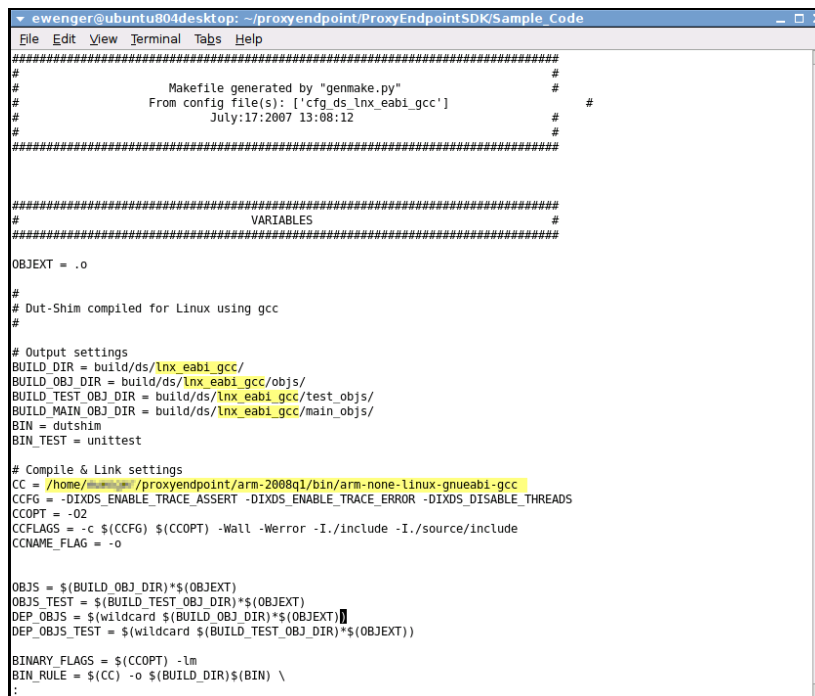


Figure 7. Modify the new makefile to use the DUT toolchain.

Step 4: Build the DUT Shim

Now that the makefile has been modified to use the cross-compiler toolchain, it's time to build the DUT shim. The new DUT shim can be built by entering 'make -f makefile_ds.eabi.gcc' at the prompt. In order to help developers build the most stable code, the makefile C flags will instruct the compiler to treat all warnings as errors. Due to differences in platforms and constant improvements to new compiler versions, some platforms may build cleanly the first time and others may require minor modifications. Please take the time to analyze any warnings to make sure that they will not affect the stability of the final binaries. An example is shown below in Figure 8. Figure 9 demonstrates the removal of the -Werror flag to allow the build to proceed with warnings.

```

wenger@ubuntu084desktop:~/proxyendpoint/ProxyEndpointSDK/sample_Code$ make -f makefile_ds.eabi.gcc
make: p build/ds/linux_eabi_gcc/
make: p build/ds/linux_eabi_gcc/objs/
gcc -o build/ds/linux_eabi_gcc/objs/endian.o -c -DIXOS_ENABLE_TRACE_ASSERT -DIXOS_ENABLE_TRACE_ERROR -DIXOS_DISABLE_THREA
ADS -D2 -Wall -Werror -I./include -I./source/include ./source/base/endian.c
gcc -o build/ds/linux_eabi_gcc/objs/error.o -c -DIXOS_ENABLE_TRACE_ASSERT -DIXOS_ENABLE_TRACE_ERROR -DIXOS_DISABLE_THREA
DS -D2 -Wall -Werror -I./include -I./source/include ./source/base/error.c
gcc -o build/ds/linux_eabi_gcc/objs/error_nix.o -c -DIXOS_ENABLE_TRACE_ASSERT -DIXOS_ENABLE_TRACE_ERROR -DIXOS_DISABLE_T
HREADS -D2 -Wall -Werror -I./include -I./source/include ./source/base/error_nix.c
gcc -o build/ds/linux_eabi_gcc/objs/error_win.o -c -DIXOS_ENABLE_TRACE_ASSERT -DIXOS_ENABLE_TRACE_ERROR -DIXOS_DISABLE_T
HREADS -D2 -Wall -Werror -I./include -I./source/include ./source/base/error_win.c
gcc -o build/ds/linux_eabi_gcc/objs/execution.o -c -DIXOS_ENABLE_TRACE_ASSERT -DIXOS_ENABLE_TRACE_ERROR -DIXOS_DISABLE_TH
READS -D2 -Wall -Werror -I./include -I./source/include ./source/base/execution.c
gcc -o build/ds/linux_eabi_gcc/objs/file.o -c -DIXOS_ENABLE_TRACE_ASSERT -DIXOS_ENABLE_TRACE_ERROR -DIXOS_DISABLE_THREA
DS -D2 -Wall -Werror -I./include -I./source/include ./source/base/file.c
gcc -o build/ds/linux_eabi_gcc/objs/memop.o -c -DIXOS_ENABLE_TRACE_ASSERT -DIXOS_ENABLE_TRACE_ERROR -DIXOS_DISABLE_THREA
DS -D2 -Wall -Werror -I./include -I./source/include ./source/base/memop.c
gcc -o build/ds/linux_eabi_gcc/objs/timevalue.o -c -DIXOS_ENABLE_TRACE_ASSERT -DIXOS_ENABLE_TRACE_ERROR -DIXOS_DISABLE_T
HREADS -D2 -Wall -Werror -I./include -I./source/include ./source/base/timevalue.c
gcc -o build/ds/linux_eabi_gcc/objs/trace.o -c -DIXOS_ENABLE_TRACE_ASSERT -DIXOS_ENABLE_TRACE_ERROR -DIXOS_DISABLE_THREA
DS -D2 -Wall -Werror -I./include -I./source/include ./source/base/trace.c
gcc -o build/ds/linux_eabi_gcc/objs/buf.o -c -DIXOS_ENABLE_TRACE_ASSERT -DIXOS_ENABLE_TRACE_ERROR -DIXOS_DISABLE_THREA
DS -D2 -Wall -Werror -I./include -I./source/include ./source/data/buf.c
gcc -o build/ds/linux_eabi_gcc/objs/buflist.o -c -DIXOS_ENABLE_TRACE_ASSERT -DIXOS_ENABLE_TRACE_ERROR -DIXOS_DISABLE_TH
READS -D2 -Wall -Werror -I./include -I./source/include ./source/data/buflist.c
gcc: warnings being treated as errors
In file included from ./include/ds/buflist.h:32:
    from ./source/data/buflist.c:22:
./include/ds/buflist.p.h: In function 'buflistNew()':
./include/ds/buflist.p.h:371: warning: dereferencing type-punned pointer will break strict-aliasing rules
./include/ds/buflist.p.h: In function 'buflistNewChunk':
./include/ds/buflist.p.h:385: warning: dereferencing type-punned pointer will break strict-aliasing rules
make: *** [build/ds/linux_eabi_gcc/objs/buflist.o] Error 1
wenger@ubuntu084desktop:~/proxyendpoint/ProxyEndpointSDK/sample_Code$

```

Figure 8. Building the DUT shim and typical build warnings.

```

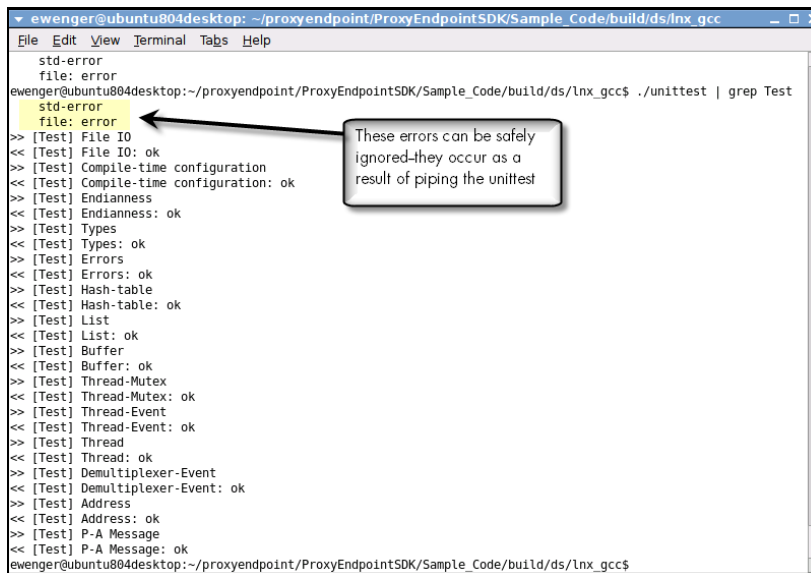
wenger@ubuntu084desktop:~/proxyendpoint/ProxyEndpointSDK/sample_Code$ cat makefile_ds.eabi.gcc
#####
#
# Makefile generated by "genmake.py"
# From config file(s): ['cfg_ds_linux_eabi_gcc']
# July 17: 2007 13:00:12
#
#####
#
# VARIABLES
#
#####
OBJEXT = .o
#
# DUT-Shim compiled for Linux using gcc
#
# Output settings
BUILD_DIR = build/ds/linux_eabi_gcc/
BUILD_OBJ_DIR = build/ds/linux_eabi_gcc/objs/
BUILD_TEST_OBJ_DIR = build/ds/linux_eabi_gcc/test_obj/
BUILD_MAIN_OBJ_DIR = build/ds/linux_eabi_gcc/main_obj/
BIN = dutshim
BIN_TEST = unittest
#
# Compile & link settings
CC = gcc
CCFG = -DIXOS_ENABLE_TRACE_ASSERT -DIXOS_ENABLE_TRACE_ERROR -DIXOS_DISABLE_THREADS
CCOPT = -D2
CCLAGS = -c $(CCFG) $(CCOPT) -Wall -Werror -I./include -I./source/include
CCNAME_FLAG = -o
OBJS = $(BUILD_OBJ_DIR)*$(OBJEXT)
OBJS_TEST = $(BUILD_TEST_OBJ_DIR)*$(OBJEXT)
DEP_OBJS = $(wildcard $(BUILD_OBJ_DIR)*$(OBJEXT))
DEP_OBJS_TEST = $(wildcard $(BUILD_TEST_OBJ_DIR)*$(OBJEXT))
BINARY_FLAGS = $(CCOPT) -ln
BIN_RULE = $(CC) -o $(BUILD_DIR)/$(BIN) \
"makefile_ds.eabi.gcc" 2858 Lines, 98787 Characters

```

Figure 9. Removing the -Werror flag.

Step 5: Install and Test the DUT Shim

Once the DUT shim has been successfully built, you will find two executables in the the 'build/ds/lx_eabi_gcc' subdirectory: *dutshim* and *unittest*. The *dutshim* binary is the DUT shim itself and the *unittest* binary will execute a number of unit tests on the *dutshim* binary to verify the program's functionality.



```

ewenger@ubuntu804desktop: ~/proxyendpoint/ProxyEndpointSDK/Sample_Code/build/ds/lx_gcc
std-error
file: error
ewenger@ubuntu804desktop:~/proxyendpoint/ProxyEndpointSDK/Sample_Code/build/ds/lx_gcc$ ./unittest | grep Test
std-error
file: error
>> [Test] File IO
<< [Test] File IO: ok
>> [Test] Compile-time configuration
<< [Test] Compile-time configuration: ok
>> [Test] Endianness
<< [Test] Endianness: ok
>> [Test] Types
<< [Test] Types: ok
>> [Test] Errors
<< [Test] Errors: ok
>> [Test] Hash-table
<< [Test] Hash-table: ok
>> [Test] List
<< [Test] List: ok
>> [Test] Buffer
<< [Test] Buffer: ok
>> [Test] Thread-Mutex
<< [Test] Thread-Mutex: ok
>> [Test] Thread-Event
<< [Test] Thread-Event: ok
>> [Test] Thread
<< [Test] Thread: ok
>> [Test] Demultiplexer-Event
<< [Test] Demultiplexer-Event: ok
>> [Test] Address
<< [Test] Address: ok
>> [Test] P-A Message
<< [Test] P-A Message: ok
ewenger@ubuntu804desktop:~/proxyendpoint/ProxyEndpointSDK/Sample_Code/build/ds/lx_gcc$

```

Figure 10. Sample output from *unittest*.

Testing with the Proxy Endpoint

Basic Test Setup

Figure 11 shows the basic network diagram for a Proxy Endpoint test. In this configuration the device under test is assumed to have only a wireless network interface which means that test traffic and test management share the wireless network. The Proxy/DUT shim communication channel has been optimized to support this type of arrangement without material impact on test performance. If the device under test does support a wired TCP/IP connection via Ethernet or USB 'gadget' interface then that can also be used to directly connect the Proxy Endpoint and DUT shim

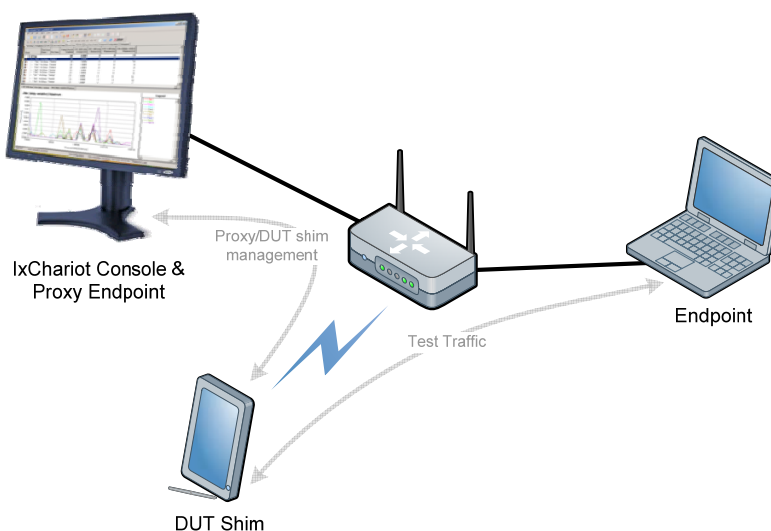
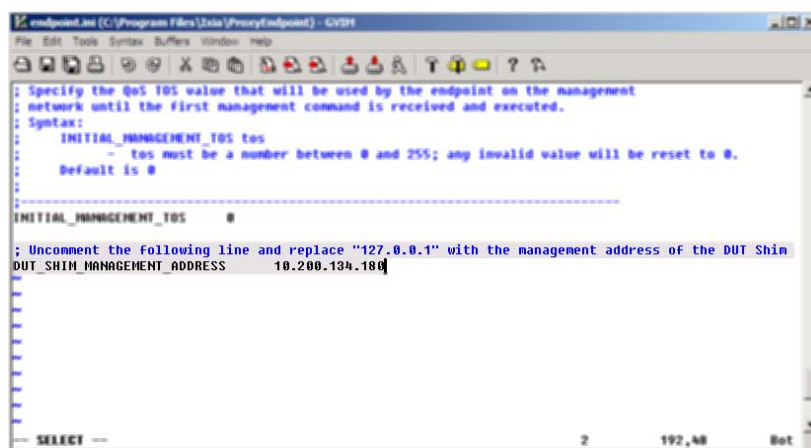


Figure 11. Typical Proxy Endpoint Test Configuration.

Step 6: Install and configure the Proxy Endpoint

The Proxy Endpoint will be installed on the IxChariot console in parallel with the existing IxChariot endpoint. The Proxy Endpoint installer can be found in the Proxy Endpoint SDK archive under 'Proxy Endpoint/peproxywin32_660.exe'. This will install the IxChariot Proxy Endpoint on your system via a standard Windows installer.

To identify the DUT shim IP address to the Proxy Endpoint, you must modify the Proxy Endpoint's endpoint.ini file located at 'C:\Program Files\Ixia\ProxyEndpoint'. The DUT shim IP address is the last entry in that file:



```
endpoint.ini (C:\Program Files\Ixia\ProxyEndpoint) - GXDI
File Edit Tools Syntax Buffers Window Help
: Specify the QoS TOS value that will be used by the endpoint on the management
: network until the first management command is received and executed.
Syntax:
INITIAL_MANAGEMENT_TOS tos
- tos must be a number between 0 and 255; any invalid value will be reset to 0.
Default is 0
-----
INITIAL_MANAGEMENT_TOS 0
: Uncomment the following line and replace "127.0.0.1" with the management address of the DUT Shim
DUT_SHIM_MANAGEMENT_ADDRESS 10.200.134.180
-----
SELECT 2 192,48 Bot
```

Figure 12. Editing the endpoint.ini file.

Since both endpoints use TCP port 10115, it is necessary to disable one of the endpoints while the other is running. This can be done via the 'Services' utility in Microsoft Windows. The easiest way to pull up this utility is to use the 'Run' option under the start menu and type 'services.msc'. It can also be found by right-clicking on 'My Computer' and selecting 'Manage'. The left hand side of the resulting window will have a 'Services' option in the tree. Figure 13 shows the two services available side-by-side. In this screenshot, the Ixia Performance Endpoint (normal endpoint) has been stopped, and the user is starting the Proxy Endpoint.

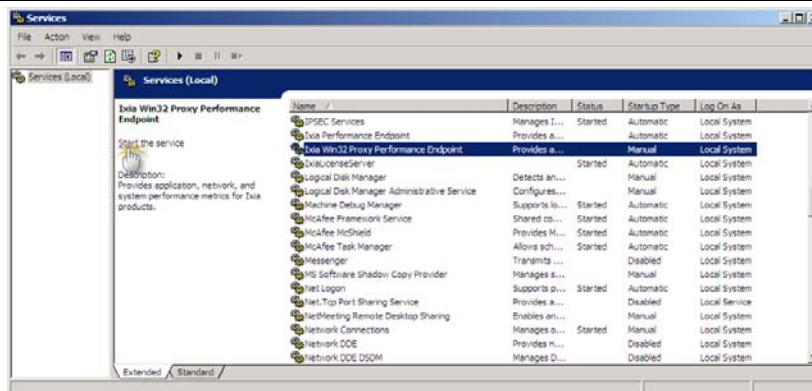


Figure 13. Starting the Proxy Endpoint service.

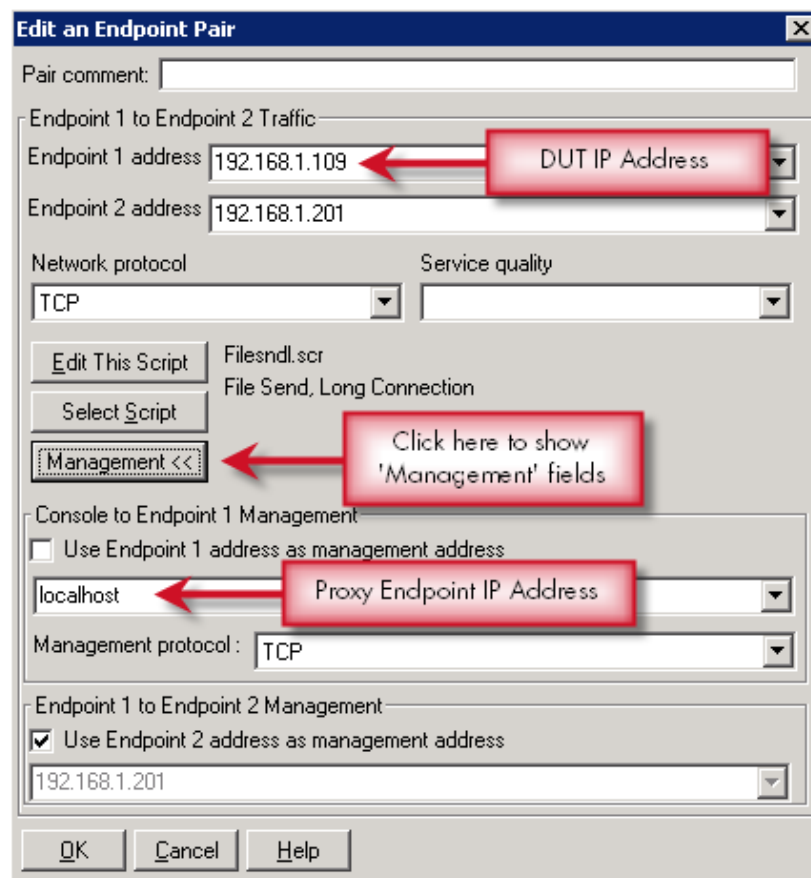
When the Proxy Endpoint and DUT shim have established communications correctly, you will see the following on the DUT:

```

root@      :~/eabi_ds$ ./dutshim
Product ..... DUT-Shim
Build time ..... Aug 11 2008 - 17:51:14
Build for platform ..... Linux
Using compiler ..... GCC [4.2]
Trace level ..... assert, error
Link-Manager: start
Link-Manager: Connected. Remote: 192.168.1.200:15837
    
```

Step 7: Run an IxChariot test with the Proxy Endpoint

The only difference between configuring a test with a native endpoint and proxy endpoint is in the selection of a management IP address. In this case, the Proxy Endpoint is managing the DUT so we'll enter the Proxy Endpoint's IP address in the E1 management field. In this case, the Proxy Endpoint is running on the console so 'localhost' is used.



Edit an Endpoint Pair

Pair comment:

Endpoint 1 to Endpoint 2 Traffic

Endpoint 1 address: 192.168.1.109 DUT IP Address

Endpoint 2 address: 192.168.1.201

Network protocol: TCP Service quality:

FilesndL.scr
 File Send, Long Connection

Click here to show 'Management' fields

Console to Endpoint 1 Management

Use Endpoint 1 address as management address

localhost Proxy Endpoint IP Address

Management protocol: TCP

Endpoint 1 to Endpoint 2 Management

Use Endpoint 2 address as management address

192.168.1.201

IxChariot Proxy Endpoint Results

Script	Direction	Native Endpoint Throughput	Proxy Endpoint Throughput
		[kbps for audio, Mbps for all others]	[kbps for audio, Mbps for all others]
filesndl	<i>dn</i>	23.163	22.836
	<i>up</i>	30.403	29.072
inquiry1	<i>dn</i>	1.623	1.596
	<i>up</i>	1.840	1.634
inquiry1-replay	<i>dn</i>	1.628	1.621
	<i>up</i>	1.826	1.629
128kbps audio (modified IPTVa)	<i>dn</i>	128.070	128.702
	<i>up</i>	127.962	128.072
IPTV 2.8 Mbps delay	<i>dn</i>	2.802	2.810
	<i>up</i>	2.815	2.806
IPTV 2.8 Mbps	<i>dn</i>	2.800	2.808
	<i>up</i>	2.806	2.806
IPTV 3.5 Mbps	<i>dn</i>	3.490	3.514
	<i>up</i>	3.496	3.496
IPTV 10 Mbps delay	<i>dn</i>	9.907	9.645
	<i>up</i>	10.074	9.970
IPTV 10 Mbps	<i>dn</i>	9.836	9.617
	<i>up</i>	9.951	9.963
IPTV 14 Mbps	<i>dn</i>	11.100	10.549
	<i>up</i>	13.974	13.975
IPTV 20 Mbps	<i>dn</i>	11.971	10.721
	<i>up</i>	19.936	19.934

Note: Measurements were conducted on a MIPS-based embedded device using Ethernet since a screen room was not available for proper wireless testing.