

Empowering Network Processors

Running A PPL Program

A Step By Step Tutorial

January 30, 2005

Copyright © IP Fabrics, Inc. 2005

IP Fabrics, Inc.
14964 NW Greenbrier Parkway
Beaverton, OR 97006
503-444-2400
503-444-2401 FAX
www.ipfabrics.com

Information in this document is furnished in connection with IP Fabrics products. No license, express or implied, to any intellectual property rights is granted by this document. This document and the software described in it are furnished under license and may only be used or copied in accordance with the terms of the license.

Copyright © 2005, IP Fabrics, Inc. All rights reserved.

Packet Processing Language™, PPL™ and PPL-VM™ are owned and copyrighted by IP Fabrics, Inc. Microsoft®, Windows® and Windows® XP are registered trademarks of Microsoft Corporation.

Linux® is a registered trademark of Linus Torvalds.

Red Hat® is a registered trademark of Red Hat, Inc.

MontaVista® is a registered trademark of MontaVista Software Inc.

Intel® and Pentium® are registered trademarks of Intel Corporation.

*Other brands, trademarks and names are property of their respective owners.

Introduction

This document is a simple step by step tutorial that guides you through the stages involved in writing, compiling, loading and running a simple PPL program.

There are two fundamental configurations generally used when writing and compiling PPL programs. In the first configuration, the Linux* version of the PPL compiler is hosted on the same Linux host containing the NPU boot server and file system. In this scenario, the PPL programs are normally created and compiled in the same file system used by the NPU. In the second configuration, either the Linux or Microsoft* Windows* version of the PPL compiler is hosted on separate, network-attached computer. In this scenario, the PPL program is written and compiled on the separate computer and the necessary files must be copied to the Linux host containing the NPU boot server and file system.

Figure 1 shows the overview of the steps needed to make a PPL program run when using the first configuration.

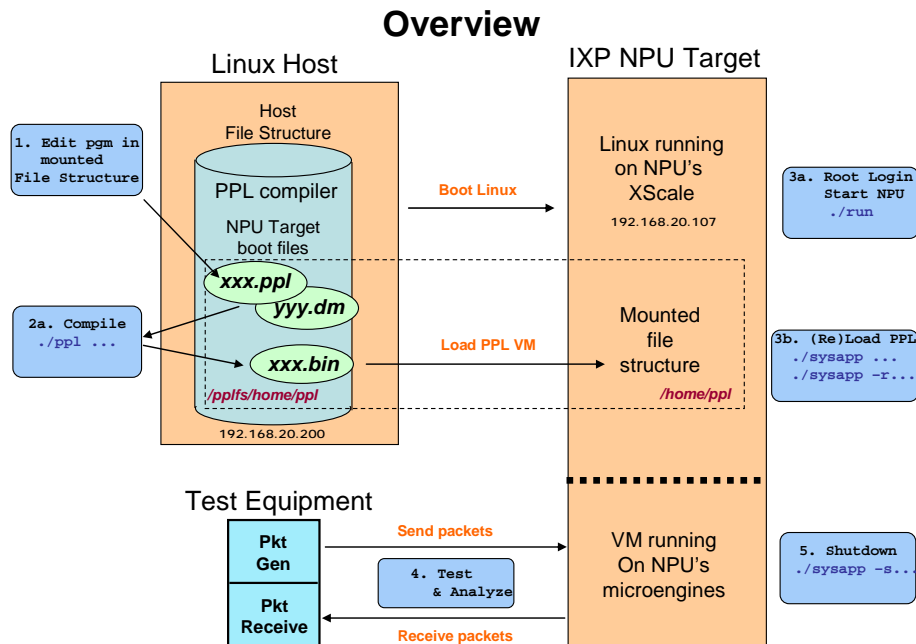


Figure 1. Steps When Using PPL Compiler Hosted on NPU Boot Host

Figure 2 shows the overview of the steps needed to make a PPL program run when using the second configuration.

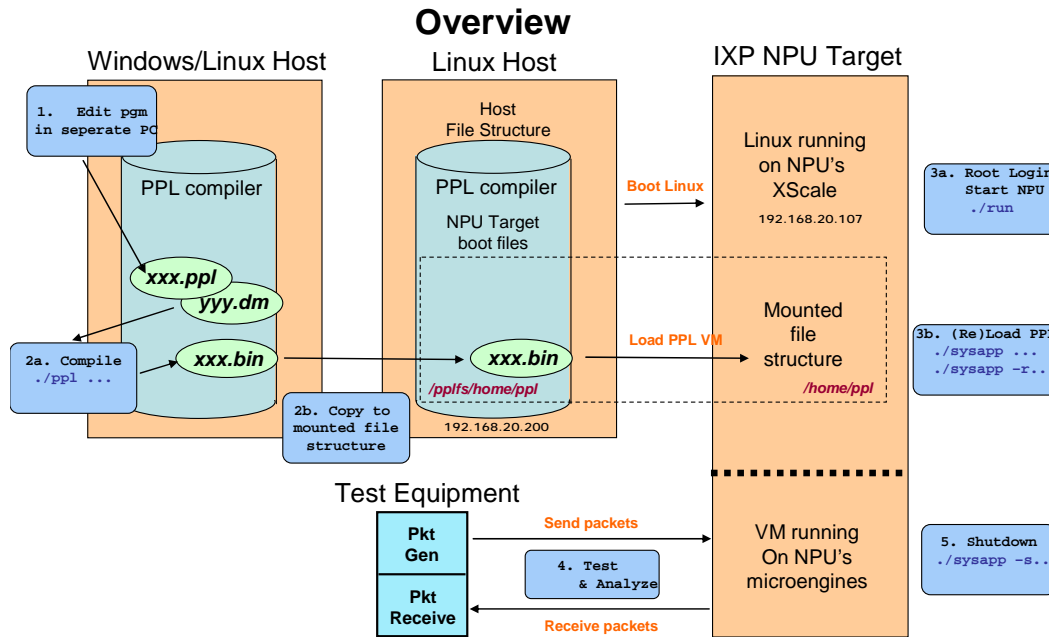


Figure 2: Steps when using PPL Compiler Hosted on Separate Computer

Table 1 contains the summary of the steps, the appropriate commands/actions within those steps and the systems on which these steps are executed. The remainder of this document explains these steps in more detail.

STEP	COMMAND/ACTION	SYSTEM
Create the PPL program	Write the PPL program. Save in ASCII text format. (use File -> Save when using Eclipse*)	Any
Compile the program	<code>./ppl ppl_filename --devicemap=devicemap_filename</code> (automatically compiled when file is saved when using Eclipse)	Compiler Host
Copy PPL Binary File to Mounted File System	<dependent on NPU Boot Server configuration> For PrPMC boot servers: <code>ftp/WinSCP/etc to 192.168.20.200 /opt/pplfs/home/ppl</code>	Compiler Host
Start the NPU	<code>./run</code>	NPU Target 192.168.20.107
Load (or reload) the PPL Binary	<code>./sysapp -b ppl_bin_file -m microcode_uof_file -t ./tables</code> Or <code>./sysapp -r -b ppl_bin_file -m microcode_uof_file -c ./deltables -t ./tables</code>	NPU Target 192.168.20.107
Analyze Results	Packet generation tools of your choice, such as Ixia* or SmartBits*.	
Shutdown	<code>./sysapp -s -c ./deltables</code>	NPU Target 192.168.20.107
View Stats (optional)	Start the <i>IPF Demo Client</i> on the Windows PC connected to the "slowport Ethernet".	Windows PC
Send Stats (optional)	Run the <i>exporter</i> <code>./exporter windows_client_ip_address</code>	NPU Target 192.168.20.107

Table 1. Summary of the Steps

Step-by-step Instructions

Step 1: Create the PPL Program

The first stage in building a PPL application is to write the PPL program itself. The discussion below refers to the *simple.ppl* program, which is listed in **Appendix I** of this document and is also included in the *ppl_samples* directory on the IPF Release CD. For examples of more complex PPL programs and Device Map files, please refer to the *ppl_samples* directory on the IPF Release CD.

Create/Write the PPL program

Use any editor (Notepad, VI, Emacs, etc.) to write your PPL program. See specific instructions below for editing your PPL program with Eclipse. Make sure to save this program in an ASCII text format with a *.ppl* file extension. The example below in **Figure 3** shows a PPL program being created in an editor. In our example we will call this file *simple.ppl*.

The *simple.ppl* program scans each incoming packet for the string "Hello World". If a match is found and the packet has a non-zero destination IP address, we forward the packet to Port 0. Any other non-zero destination IP address packet is forwarded to Port 1. A packet with destination IP address of zero is dropped.

```

# connections section
# =====

# array section
# =====

# =====
# event section
# =====
EVENT (0)

# rule section
# =====

# for this rule to be true, set IP DEST should be non zero
# and the packet must contain the string "Hello World", if
# so forward to Port 0

RULE NE(IP_DEST, 0) SCAN("Hello World") FORWARD(0) STOP

# otherwise if non zero destination IP forward packet to Port 1

RULE NE(IP_DEST, 0) FORWARD(1) STOP

# else just drop the packet

RULE DROP

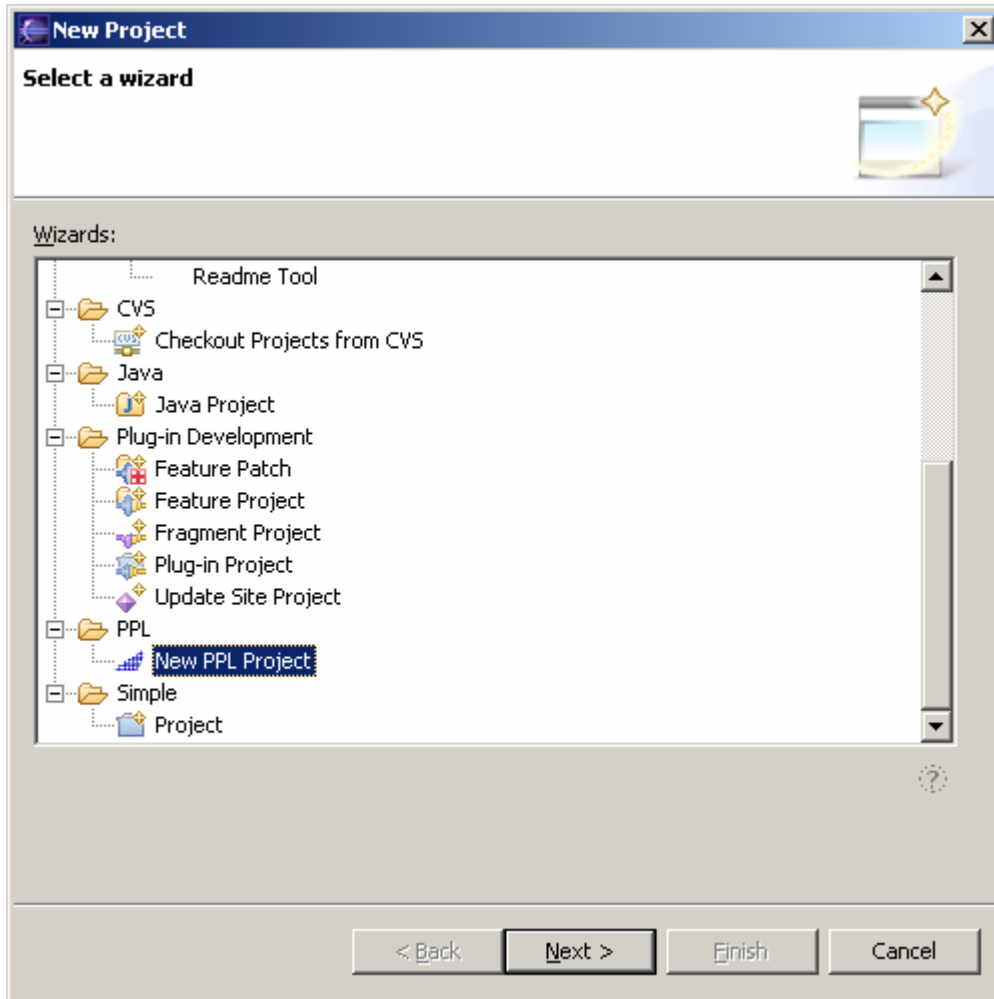
```

Figure 3. Editing the PPL program

Create/Write the PPL program using Eclipse

See **PPL Installation Guide** for Eclipse installation.

PPL programs should be created and stored under PPL Projects. A PPL Project is created by invoking the PPL Project Wizard via **File > New > PPL > New PPL Project** under Eclipse.

**Figure 4. PPL Wizard**

Once a PPL Project folder is created, create the member files by selecting **File > New > File**. PPL program files should have an extension of `.ppl` and Device Map files should have an extension of `.dm`. Use the PPL and Device Map editors to create or change your PPL program.

Setup the appropriate Eclipse “editors” by setting the file associations as follows:

- Access the file associations page via **Window > Preferences > Workbench > File Associations ...** from the Eclipse Menu. See **Figure 5** below.

- Add a `.dm` and `.ppl` file type and add associated editors for each of Device Map Editor and PPL Editor as the default, respectively.

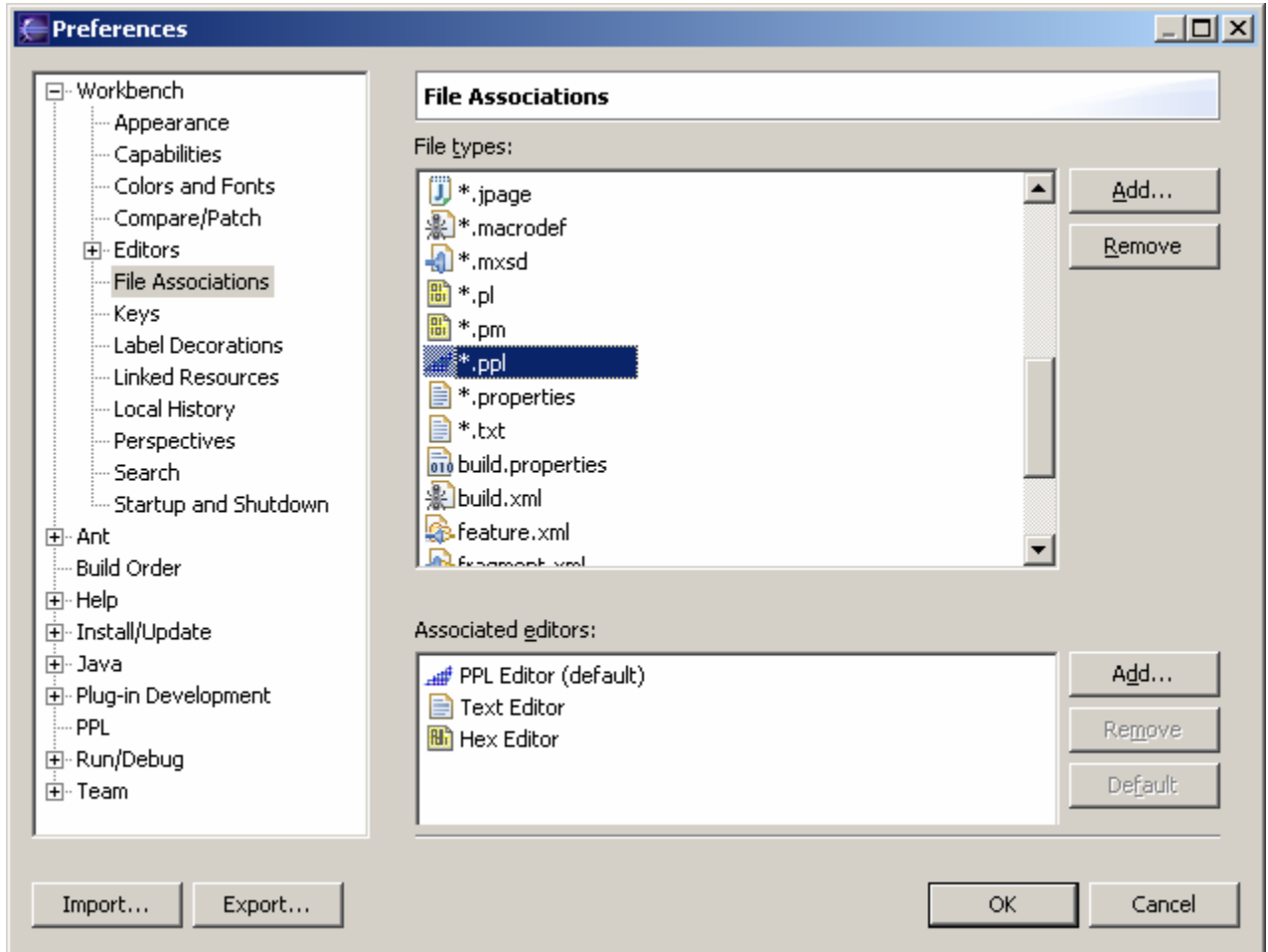


Figure 5. File Association Configuration

Step 2: Compile the PPL Program

Compiling the PPL program from the Linux Command Line

The next step is to compile the PPL program using the PPL Compiler. This step assumes that the installation and configuration of the PPL suite has already been completed. Please refer to the *Installation Document* for the installation of the PPL Compiler on a Pentium Linux machine.

Copy the PPL program along with the Device Map needed for compilation to the subdirectory in the Pentium Linux machine in which the PPL Compiler has been installed (any subdirectory will do so long as the files along with their paths are specified correctly to the PPL Compiler).

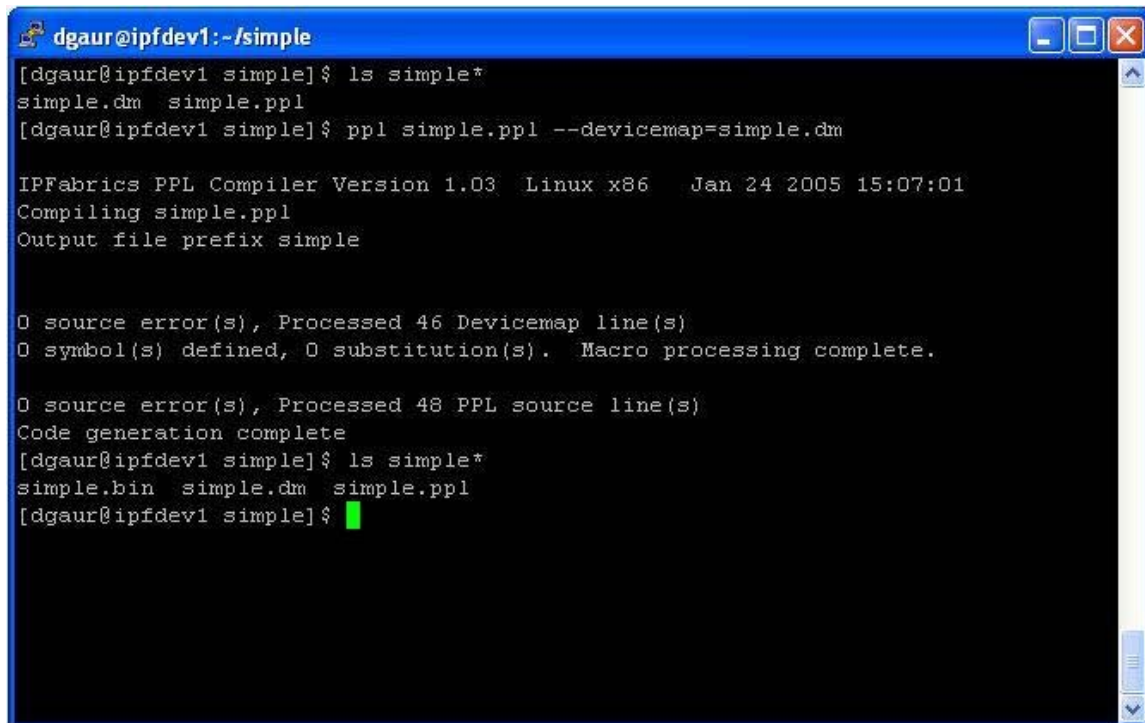
Next compile the PPL program with appropriate Device Map by using the following command at the prompt in the subdirectory.

```
./ppl ppl_file.ppl --devicemap=devicemap_file.dm
```

For Example:

```
./ppl simple.ppl --devicemap=simple.dm
```

Upon successful compilation of the PPL program, the PPL Compiler emits a *.bin* file with the same name as the PPL program. **Figure 6** shows the successful compilation of *simple.ppl* to *simple.bin* using the Device Map file *simple.dm*.



```

dgaur@ipfdev1:~/simple
[dgaur@ipfdev1 simple]$ ls simple*
simple.dm  simple.ppl
[dgaur@ipfdev1 simple]$ ppl simple.ppl --devicemap=simple.dm

IPFabrics PPL Compiler Version 1.03  Linux x86   Jan 24 2005 15:07:01
Compiling simple.ppl
Output file prefix simple

0 source error(s), Processed 46 Devicemap line(s)
0 symbol(s) defined, 0 substitution(s).  Macro processing complete.

0 source error(s), Processed 48 PPL source line(s)
Code generation complete
[dgaur@ipfdev1 simple]$ ls simple*
simple.bin  simple.dm  simple.ppl
[dgaur@ipfdev1 simple]$

```

Figure 6. Successful compilation of the PPL program

If the Compiler detects an error in the PPL program, it will report the error along with the line number of the PPL program on which the error occurred. If the Compiler does report an error, correct it and repeat **Step 2** to produce a *.bin* file.

Compiling PPL program using Eclipse

When either the *.ppl* or *.dm* file within a project is saved, Eclipse automatically invokes the PPL Compiler to generate the *.bin* file, provided the “Build automatically” checkbox is selected under **Window > Preferences > Workbench** (this is the default setting). Compilation errors are displayed in the Problems view and the source text is annotated with a marking on the source line containing the error. The compiler may not detect some errors until the start of a new statement, so that the error marker may appear on a line subsequent to where the actual error occurred.

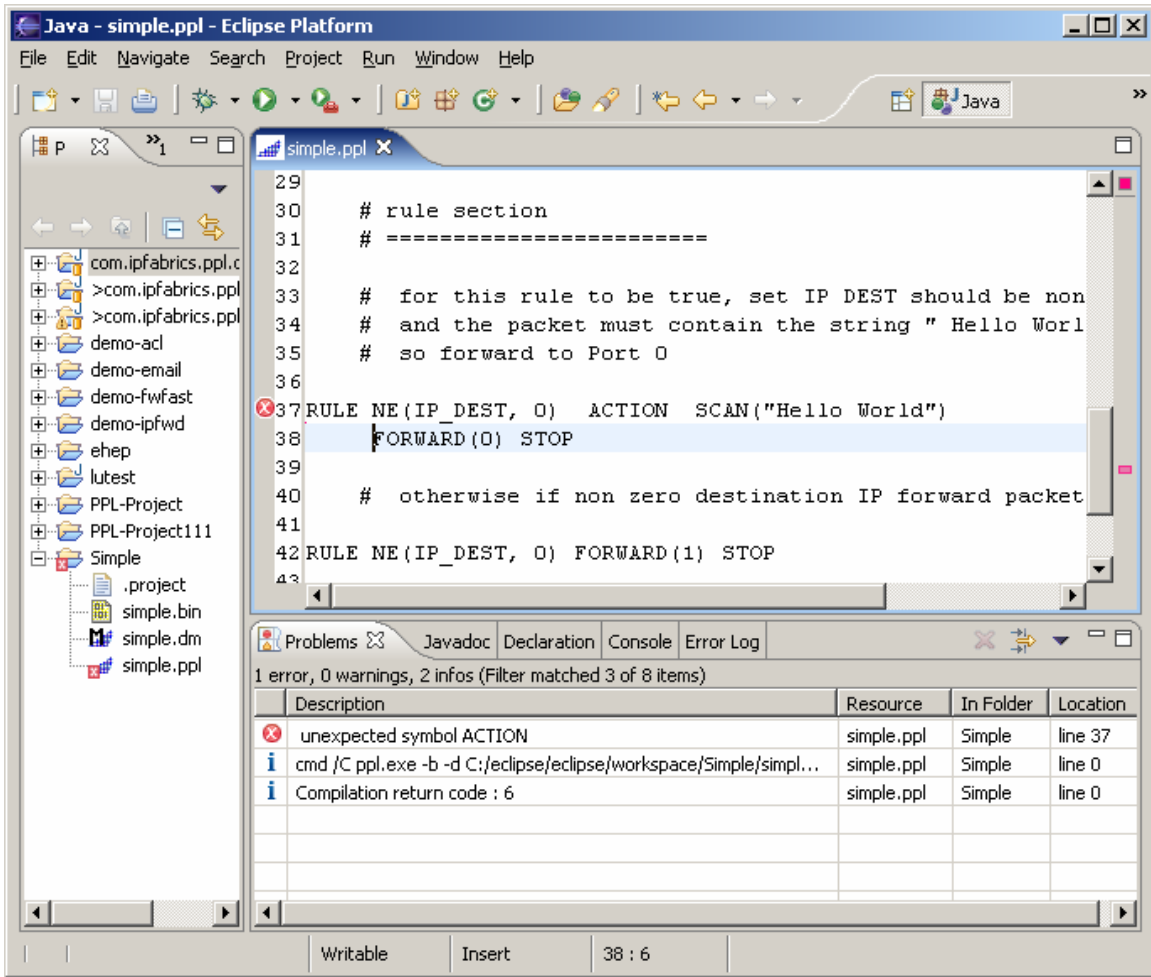


Figure 7. Indicating errors in the PPL program

The PPL compiler (*ppl* under Linux or *ppl.exe* under Windows) and final code generator (*pplg* or *pplg.exe*) must be in your path. In Windows, use the Control Panel (**System > Advanced > Environment Variables**) to include these programs in your path. If you used the default Windows installer, the compiler tools are located in *<Program Files>\ppl*.

Step 3: Copy the Binary File to the NPU

Once compiled, the PPL binary file (*.bin*) must be copied to the NPU target's file system. Typically, the NPU target's file system is hosted either on the embedded PrPMC attached to the NPU blade or on a user-supplied boot server.

In Linux, the PPL compiler will emit the *.bin* file in the directory in which it was invoked. In Windows, the binary file can be found on the Windows host under *<eclipse root directory>/<PPL-Project name>*. Copy the *.bin* file to the file system mounted on the NPU target, which should be */opt/pplfs/home/ppl* on the PrPMC or boot server. Since the PrPMC only offers network connectivity, use FTP (or a similar utility) to copy the file (the PrPMC's static IP address is 192.168.20.200).

Step 4: Load the Compiled PPL Program using the SysApp

Connect to the NPU target using telnet (or a similar utility). Log on as *root*. The PPL suite and compiled *.bin* file should appear on the NPU target under the */home/ppl* directory.

First, we need to execute the *run* script file if this is the first time we are running the System Application (*SysApp*) since reboot. This will setup all the drivers that are needed by the IXP hardware. Use the following command within the */home/ppl/* subdirectory at the prompt.

```
./run
```

Next we need to load the IXP microengines using the *SysApp* application as well as setup the *route* table. Use one of the following commands at the prompt in the */home/ppl/* subdirectory.

If you are running the *SysApp* for the first time since reboot:

```
./sysapp -b ppl_bin_file -m microcode_uof_file -t ./tables
```

For Example:

```
./sysapp -b simple.bin -m PPL_VM_2800.uof -t ./tables
```

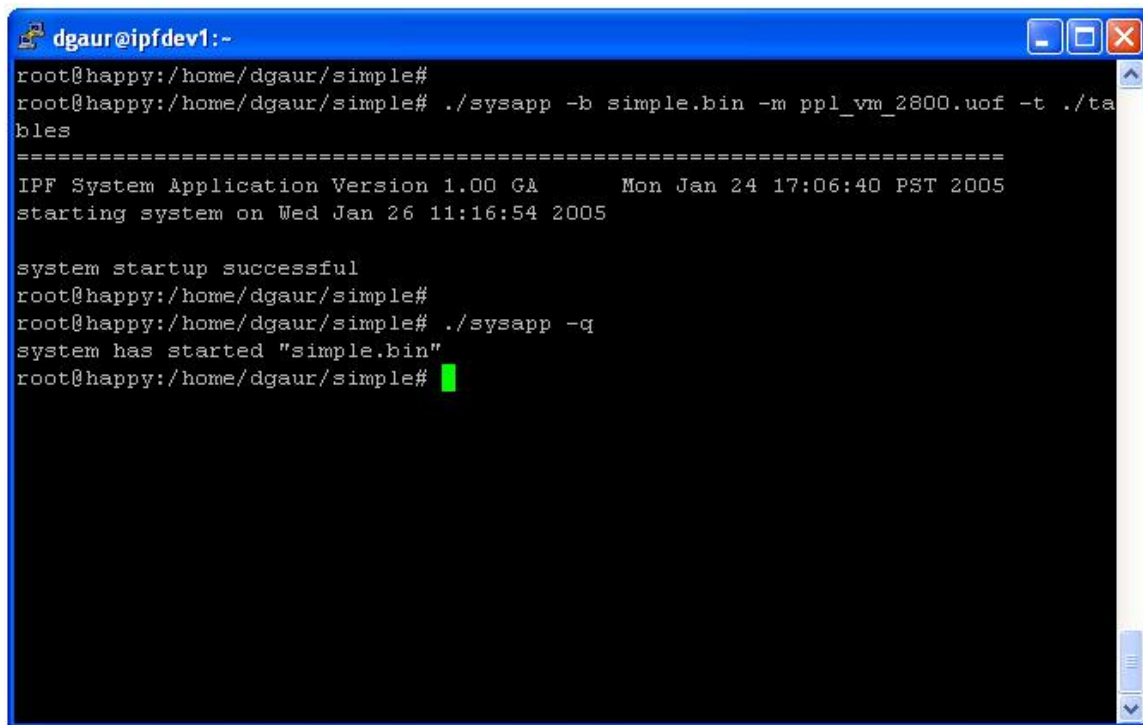
Or, if you are just restarting the PPL VM (without a reboot) to load a new *.bin* file:

```
./sysapp -r -b ppl_bin_file -m microcode_uof_file -c ./deltables -t ./tables
```

For Example:

```
./sysapp -r -b simple.bin -m PPL_VM_2800.uof -c ./deltables -t ./tables
```

Figure 8 shows *simple.bin* being successfully loaded by the *SysApp*.



```
dgaur@ipfdev1:-
root@happy:/home/dgaur/simple#
root@happy:/home/dgaur/simple# ./sysapp -b simple.bin -m ppl_vm_2800.uof -t ./tables
=====
IPF System Application Version 1.00 GA      Mon Jan 24 17:06:40 PST 2005
starting system on Wed Jan 26 11:16:54 2005

system startup successful
root@happy:/home/dgaur/simple#
root@happy:/home/dgaur/simple# ./sysapp -q
system has started "simple.bin"
root@happy:/home/dgaur/simple#
```

Figure 8. Successful loading of the PPL *.bin* file

Any errors in the loading will be reported to the `/var/log/syslog` file. The `sysapp` reports the successful/unsuccessful completion of the loading of the `.bin` file.

After the successful loading of the `.bin` file, the microengines will be up and running. The PPL VM is now ready to receive and process packets.

The `tables` and `deltables` script files that we specified in the commands are required for setting up the `route` table. For more information on the `tables` script, `deltables` script and the setup of the `route` table, please refer to Appendix II of this document.

Step 5: Viewing the results of the PPL program

If you are using a packet generator to test your PPL program, then the results can be verified by monitoring the traffic sent by the PPL program.

Additional methods such as logging will be included in the future releases.

As an optional step, we can run the included Exporter (`/home/ppl/`) application. This allows a Windows client application (included in the `Demo` directory of the Release CD) to graphically display the activity within the Microengines.

Start the client on the Windows client machine by running the IP Fabrics Demo Client 01 application (the `IP Fabrics Demo Client 01.pdb` file must be in the same directory as the `.exe`). This will create a GUI window which displays the Microengine activity within the PPL VM.

Then start the Exporter on the NPU by using the following command at the prompt in the subdirectory where the `exporter` is installed (`/home/ppl/`).

```
./exporter client_ip_address
```

For Example:

```
./exporter 192.168.0.8
```

Figure 9 shows the IPF Demo Client application displaying Microengine activity for a PPL program.

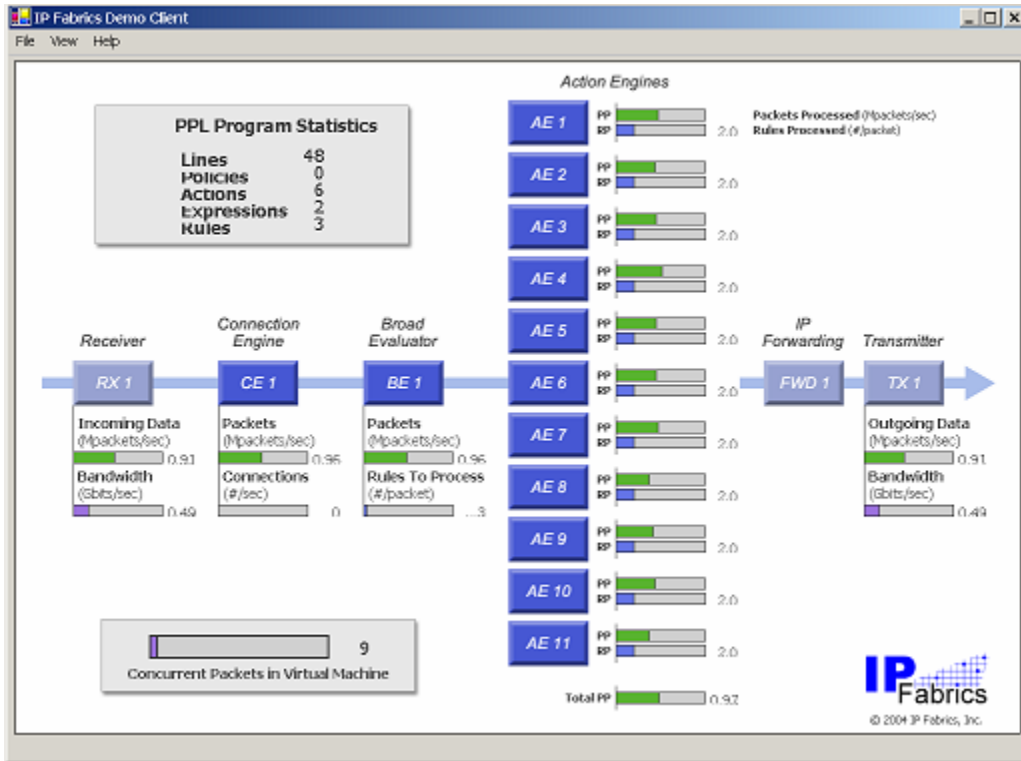


Figure 9. IPF's Windows GUI Client

The IP Fabrics Demo Client 01 application included in the Release CD will not display the statistic information in the PPL Program Statistics Box within the GUI at this time. This will be included in a future release.

Step 6: Shutting down SysApp

To shutdown the SysApp application, use the following command within the `/home/ppl/` subdirectory at the prompt.

```
./sysapp -s -c ./deltables
```

Appendix I: Sample PPL Program

Listing of the simple.ppl file

```

# =====
# # date      description                                     by
#
# 001 06.21.04 simple Hello World scan program demonstration IPF
# =====
# file: simple.ppl
#
# description: contains a simple ppl program to demonstate the
#               writing, compiling, loading and running of a
#               ppl program
# =====

# define/constant section
# =====

# policy section
#=====

# connections section
# =====

# array section
# =====

# =====
# event section
# =====
EVENT(0)

# rule section
# =====

# for this rule to be true, set IP DEST should be non zero
# and the packet must contain the string " Hello World", if
# so forward to Port 0

RULE NE(IP_DEST, 0) SCAN("Hello World") FORWARD(0) STOP

# otherwise if non zero destination IP forward packet to Port 1

RULE NE(IP_DEST, 0) FORWARD(1) STOP

# else just drop the packet

RULE DROP

# =====

```

Listing of the simple.dm file

```

devicemap
# model clock
npu(2800, 1400)
#               low high
available_processors(0, 15)
# rx      be      sl      ae      pe      fwd      tx
ppl_processors(ER(10%), BE(10%), CE(10%), AE(40%), PE(10%), I4(10%), ET(6%))

packet_mem(DRAM,16384)
log_mem(DRAM,4096)
journal_mem(DRAM,100)

```

```

connections_mem(SRAM)
associations_mem(SRAM)
array_mem(DRAM)

#  lpn dir      type  sport mtu          smac          pm sfc rfc  cntrl cport
link(0, INOUT, GE_ON_SPI, 0, 1500, 00:07:e9:2d:33:88, 1, 0, 0, IXF1104-0, 0)
link(1, INOUT, GE_ON_SPI, 1, 1500, 00:07:e9:2d:33:89, 1, 0, 0, IXF1104-0, 1)
link(2, INOUT, GE_ON_SPI, 2, 1500, 00:07:e9:2d:33:8a, 1, 0, 0, IXF1104-0, 2)
link(3, INOUT, GE_ON_SPI, 3, 1500, 00:07:e9:2d:33:8b, 1, 0, 0, IXF1104-0, 3)
#link(4, INOUT, GE_ON_SPI, 4, 1500, 00:07:e9:2d:33:8c, 0, 0, 0, IXF1104-1, 0)
#link(5, INOUT, GE_ON_SPI, 5, 1500, 00:07:e9:2d:33:8d, 0, 0, 0, IXF1104-1, 1)
#link(6, INOUT, GE_ON_SPI, 6, 1500, 00:07:e9:2d:33:8e, 0, 0, 0, IXF1104-1, 2)
#link(7, INOUT, GE_ON_SPI, 7, 1500, 00:07:e9:2d:33:8f, 0, 0, 0, IXF1104-1, 3)

#
#          flow_ordering, arp
packet_control(0, 1)

# 1 = IP version not v4 or v6
# 2 = IPv4 header len < 5, total len < 4 * IHL, invalid checksum
# 3 = if IP packet, IP_PROT >= 136
# 4 = IP packet size inconsistent with Ethernet frame size
# 5 = improper source address
# 6 = improper destination address
# 7 = source IP address = destination IP address
# 8 = malformed TCP packet
# 9 = malformed UDP packet
# 10 = malformed ICMP packet
# 11 = bad fragment
reject_malformed(2,3,4,5,6,7,8,9,10,11)

```

Appendix II: Table Scripts

The *tables* and *deltables* scripts use the command line programs *l2config* and *rconfig*. All these scripts are present in the */home/ppl/* subdirectory.

USAGE: *l2config* <cmd> <"args">

```

where <cmd> <"args"> pair can be one of the followings:
addV4EthEntry "l2Index ipAddr DestMAC SourceMAC name"
addV6EthEntry "l2Index IPv6Addr DestMAC SrcMAC name"
addV4L3Info "l2Index ipAddr name"
addV4AtmEntry "l2Index ipAddr vpiVci vcq llcSnap name"
deleteL2Entry "l2Index name"
clearL2Info "l2Index name"
dumpL2tm "name"

```

USAGE: *rconfig* <cmd> <"args">

```

where <cmd> <"args"> pair can be one of the followings:
addNextHop "nextHopID bladeID l2Index portID mtu flags ipAddr l2IndexType"
delNextHop "nextHopID"
addRoute "prefix netmask nextHopID"
delRoute "prefix netmask"
dumpRtm
lookupRoute "ipaddr"
purgeRoutes

```

l2config manipulates the L2 table which is used by the Forwarder and Transmit microblocks. *rconfig* manipulates the route table and nexthop table which are used by the Forwarder microblock.

The "*l2index*" used in most of the *l2config* commands is simply an index into the L2 table. This same index is used in the "*rconfig addNextHop*" command so you need to specify the L2 entry

first, then the “*nexthop*” associated with it. Conversely, the “*nextHopID*” is also an index, but this time into the *nexthop* table. The “*rconfig addRoute*” command uses the ID, so you need to have the “*nexthop*” entered first before adding a route.