Title	Debugging J-Link Applications.						
Product	Pro/ENGINEER	**Module**	J-Link	**TPI ID**	34380	**Created**	29-JUN-99
Workstation	All			**Reported In Release**	2000i	**Reported In Datecode**	1999140
SPR	None			**Resolved In Release**		**Resolved In Datecode**	

```
Description
-----------
This document outlines some techniques that can be used to debug J-Link
Applications.

Alternate Technique
-------------------
See resolution below.

Resolution
----------

Pro/ENGINEER starts the Java Virtual Machine (VM) with the following command line
arguements:

<jre command line> com.ptc.pfc.Implementation.Starter <host> <port>

com.ptc.pfc.Implementation.Starter is the mandatory name of the J-Link class that
initializes J-Link and establishes connection between the VM and Pro/ENGINEER.

<host> is the host name of the machine where the application is running, and <port>
is a system-assigned port number allocated by Pro/ENGINEER for connection with the
VM. Note: do not use "localhost" or "loghost" as a host name, it will not work due
to a temporary limitation in the socket library.

<jre command line> varies by platform.
The defaults for supported platforms are:

sun4_solaris:
  jre -native

sgi_elf2:
  java -native -o32 -nojit

sgi_mips4:
  java -native -n32 -nojit

alpha_unix:
  jre -native -nojit

ibm_rs6000:
```

```
  jre -classpath "$CLASSPATH"

hp8k: NOTE! ON hp8k ONLY J-Link RUNS IN GREEN THREADS MODE
  jre

i486_nt, i486_win95, alpha_nt:
  jre -classpath "$CLASSPATH"

SGI Users note: A full JDK install is required even if no Java programs are being
compiled on the target machine, while on other platforms JRE is enough to run
programs.

In order to debug a J-Link program with jdb, users *must* be running under a debug
VM. This can be a pure debug VM (java_g or jre_g), to which the user can remotely
connect from jdb, or it could be jdb itself started directly from Pro/ENGINEER.

In order to start a debug VM, the user must change the VM startup command that
Pro/ENGINEER executes:

  setenv         PRO_JAVA_COMMAND         "<jre_g command line>   -
debug     com.ptc.pfc.Implementation.Starter"

<jre_g command line> is the same as <jre command line> above, except that it must
start jre_g or java_g instead of jre or java. Note the addition of -debug parameter.
That parameter causes the VM to print a "password" that allows jdb to remotely
connect to the VM. Upon startup, the VM will print something like:

  Agent password=t8gav

The user can then start jdb:

  jdb -password t8gav

It will connect to the running VM. The user can then set breakpoints, print out
variable values and use any available jdb commands.

NOTE: jdb *must* have the same CLASSPATH as the user application in order to work
correctly. When a J-Link application starts up, J-Link prints out
the application's CLASSPATH to standard output.

NOTE: On Windows platforms these messages output by Pro/ENGINEER are not seen in the
MSDOS window from where the Pro/ENGINEER startup command was executed from. users
are directed to use the syntax

DOS> proe2000i.bat > output.txt

for starting Pro/ENGINEER which will then capture the messages in the output.txt
file.

All user classes *must* be compiled with debug information (javac -g) in order for
jdb to show source code and values of variables.

On all platforms except hp8k, J-Link loads a native library for communication with
Pro/ENGINEER.
The name of the library on the different platforms is as follows:

sun4_solaris, sgi_elf2, sgi_mips4, alpha_unix:
  libjavaxportmt.so

ibm_rs6000:
  libjavaxportmt.a

i486_nt, i486_win95, alpha_nt:
  javaxportmt.dll

If a debug VM is used, the VM will automatically append "_g" to the end of the name
of the library, and there is no way to prevent it. For instance, on Solaris the
debug VM will try to load libjavaxportmt_g.so, and on NT javaxportmt_g.dll. The
default installation will not provide that library. That library must be the same as
```

```
the installed javaxport library.
There are two ways to overcome this limitation. One way is to make a copy or a link
in $PRO_DIRECTORY/$PRO_MACHINE_TYPE/lib. Another way is to create a copy or a
symbolic link in a local directory and add that local directory to the beginning of
the shared library path. The shared library path environment variable on different
platforms is:

sun4_solaris, sgi_elf2, alpha_unix:
  LD_LIBRARY_PATH

sgi_mips4:
  LD_LIBRARYN32_PATH

ibm_rs6000:
  LIBPATH

hp8k (provided for completeness):
  SHLIB_PATH

i486_nt, i486_win95, alpha_nt:
  PATH

On UNIX, the elements of the path are separated with ":" (colon), on Win32 with
";" (semicolon).

Another way is to start jdb right from Pro/ENGINEER. Jdb has no argument -nojit: (it
never runs with JIT). Other than that, its command line is the same as the above
command lines for java and jre, except the command is jdb instead of java or jre.

Inorder to establish the port number used for the connection follow the technique
illustrated below: Write a script that echoes port# and then runs jdb.

For sgi_elf2:

cat > run_jdb << END_SCRIPT
#!/bin/csh -f
echo Starter arguments: $1 $2
jdb -native -o32 com.ptc.pfc.Implementation.Starter
END_SCRIPT
chmod +x run_jdb
setenv PRO_JAVA_COMMAND run_jdb

The output will look like this:

  Starter arguments: oldpug 2386
  Initializing jdb...
  Thread-4[1] run com.ptc.pfc.Implementation.Starter oldpug 2386

Please note that Pro/ENGINEER times-out after 20 seconds by default when waiting for
the VM to start up and connect to Pro/ENGINEER. If this timeout functionality needs
to be disabled:

  setenv         PRO_JAVA_CONNECT_TIMEOUT         0

Also note that setting the PRO_JAVA_COMMAND environment variable gives a great deal
of freedom.
Customers can actually use *any* JDK 1.1 compliant VM, although PTC dosen't
officially support VM's other than the ones that have been tested with J-Link (a
list is included in the JLink documentation). Using a different VM like for instance
Symantec's may allow the customers to utilize a more friendly debugger than jdb,
like the one in Visual Cafe. Users will have to get a command line version of the
debugger vendor's VM, find out what its command line arguments are, and set
PRO_JAVA_COMMAND accordingly.

One important thing to keep in mind is that except on hp8k, the VM must run with
native threads, otherwise the native communication library will not work. Hence the
switch "-native" on platforms that have both native and green threads. There is a
way around though. If you add "-DcipPureJava" to the jre/java/jdb command line, J-
Link will use a Java communication library instead of the native library, and that
```

```
should work with a green thread VM on any platform. It is especially useful with
third-party VM's that may not support native threads. Please note that -DcipPureJava
is *unsupported*, it's just an undocumented feature that may prove helpful in some
cases for highly advanced users, but may change or disappear altogether at any time.
```